Multi-label Text Classification With Self-Tuning using Random Search and Differential Evolution

Amit Kanwar
Master of Computer Science
North Carolina State University
akanwar2@ncsu.edu

Gaurav Joshi
Master of Computer Science
North Carolina State University
gjoshi@ncsu.edu

Rahul Gutal
Master of Computer Science
North Carolina State University
rmgutal@ncsu.edu

ABSTRACT

Data miners have multiple applications in many problems in research areas including mathematics, cybernetics, genetics and marketing, and there can be multiple algorithms that can be used with different sets of parameters to solve the same problem. One of the problems where data miners can be used is Classification of text into multiple classes known as a “Multi-label text classification” problem. In this paper, we talk about the approach used by most of the papers such as paper[3] to perform multi-label text classification on news articles and discuss about various important criteria that should be taken care while developing data miners, followed by an improved implementation of currently used most common techniques by implementing self-tuning of parameters of the learner algorithm.

KEYWORDS

Classification, Multi-label text classification, Parameter tuning, Random Search, Differential Evolution

1 INTRODUCTION

Data mining is the computing process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics and database systems[16]. It is an interdisciplinary subfield of computer science[17] where intelligent methods are applied to extract data patterns. The purpose of a good Data Miner is to present a tool to convert the data to useful information. But, it looks like most of the data miners that are being developed are focused on only getting good accuracy on a certain dataset for a certain problem, without considering other important factors.

Many research papers have reported methods to improve the results of classifiers but most of them such as paper[28] are focused on using different versions of the classification algorithm and using various preprocessing methods over the data to get improved results. According to us, self tuning of parameters of the used classifiers is a very important criteria to consider while writing efficient data miners. We aim to improve the existing commonly used techniques to solve multi-label text classification problem, by implementing self-tuning over them.

Research Questions

- **RQ1:** Can we get better results with self-tuning on classifiers?

Result 1

The tuning of parameters of learners used, improved the results for most of the classifiers for multi-label dataset and for some of the classifiers for multi-class dataset.

- **RQ2:** Are the results significantly better than the solutions that ignore self-tuning?

Result 2

We compared the results using the performance measures and significance tests, and found out that the results were significantly better after tuning.

- **RQ3:** How well do the simpler classification algorithms such as Naive Bayes and Logistic Regression perform as compared to more complex ones such as Support Vectors?

Result 3

While the performance of Naive Bayes and Logistic Regression improved a lot after tuning, Support Vector classifiers still performed better. However, Logistic Regression after tuning was able to match the performance of Support Vector without tuning as confirmed by the rankings obtained by significance tests.

In summary, the contributions of this paper are:
• Improvement in performance after tuning as compared to without tuning.
• A complete comparison and ranking of 5 classifiers based on the results on 2 of the most widely used datasets.
• Implementation of parameter tuning using Differential Evolution and Random Search techniques that can be used for any dataset.

The rest of the paper is structured as follows: Section 2.1 gives an overview of the problem of Multi-label text classification, Section 2.2 describes various criteria and selection of the Key criteria that should be taken care while building data miners, Section 2.3 discusses about Related Work, Section 2.4 describes our proposed work with details of classifiers and tuning methods used. Section 3 describes our experimental setup with details of datasets, data sampling techniques and evaluation criteria used. Section 4 describes the results and answers to the research questions. Section 5 validates the results presented in Section 4 while Section 6 includes our conclusions.

2 BACKGROUND AND MOTIVATION

2.1 Multi-label Text Classification

Classifying the textual data into multiple categories or labels is called text classification. If there are more than categories to decide, then the problem is multi-class text classification and if every data point can belong to more than 1 category then that problem is known as a "Multi-label text classification" problem. A multi-label classification problem arises in various domains such as labeling multimedia resources such as images, videos and audio files, genetics/biology and text classification such as emails, documents, reports, patents and web pages.

Multi-label text classification is an important problem in data science and has a multitude of business use cases such as Automated Ads suggestion using classification of web pages, Automated Gene sequencing, Automated review categorizer and Medical diagnosis to find multiple ailments at the same time[6].

2.2 Model Criteria

We present certain criteria[9] that should be considered to develop very useful understandable and adaptable data miners.

Model readability

Miners are based on models that take the data as input and give the output that is used for prediction. Anyone who has to explain the output to the business user or defend a conclusion against a critical audience should have a readable model output that is easy to explain[9]. Output of some learner algorithms such as Naïve Bayes classifier is very complicated and not easily explainable thus, critical questions can't be asked or answered using that output. But there are some learners such as Decision Trees which have a nice representation of the output which can help the business user understand how the learner is being used for the decisions.

Model readability is very useful when presenting the outputs to the business user, the readability of a model can increase its usability but, achieving this is very difficult because of statistical complexity of some learner.

Actionable conclusions

The conclusions drawn from the model, depend on various attributes and parameters of the model. Once we present the model and its output to the business user, the model should also be plannable, i.e. based on the conclusions derived from the current model, we must know which parameters should be changed in the learner model to get a different result or some improvement based on the requirements of the business user.

This is very important because the requirements vary in different domains and it gives the flexibility to make changes to get the best result for a particular problem based on user requirements.

Learnability and Repeatability of the results

The model should use less CPU time and disk/RAM so that it is fast to train/test. This means the model is easy to learn and repeat. During the process of building the model, the model has to be run multiple times with some parameter tuning or on large volumes of data, so, it's not advisable to have a slow learner because for making any changes, you have to wait for the model to train and test and if it's taking hours to do so on a relatively small dataset, it will waste the time and resources just to later find out that a small tweak was required for a better result.

This is very important while comparing performance of different learners because we have to run the models multiple times. We can achieve almost similar performance with simpler and faster models. Also, when running the models on cloud technologies, a fast model would save money. Mini batch k-means can be used to reduce the space-complexity for a clustering problem.

Multi-goal reasoning

Multi-objective optimization is an integral part of optimization activities and has a tremendous practical importance, since almost all real-world optimization problems are ideally suited to be modeled using multiple conflicting objectives. The classical mean of solving such problems were primarily focused on scalarizing multiple objectives into a single objective, whereas the evolutionary means have been to solve a multi-objective optimization problem as it is [19].

The miner should be able to reason its performance based on its objective or goal function. In many problems, there are multiple objective functions and the performance of a model on one goal might affect its performance on some other goal so, the miner should be able to reason with all the objectives and find a solution that does well overall, considering all the goals.

Anomaly detection

Anomaly detection is a technique used to identify unusual patterns that do not conform to expected behavior, called outliers. It has many applications in business, from intrusion detection (identifying strange patterns in network traffic that could signal a hack) to system health monitoring (spotting a malignant tumor in an MRI scan), and from fraud detection in credit card transactions to fault detection in operating environments [2].

Most current anomaly detection techniques only consider the content of the data source i.e. the data itself, without concern for the context of the data. For example, a sensor reading may determine that a particular electrical box is consuming an abnormally high amount of energy. However, when viewed in context with the location of the sensor, current weather conditions and time of the year, it is well within the normal bounds [20].

Not all learners can be used for anomaly detection. K-means can be used to make clusters of data points and data points falling outside these groups can be classified as anomalies.

Incremental

In incremental learning, input data is continuously used to extend the existing model's knowledge i.e. to further train the model. It represents a dynamic technique of supervised and unsupervised learning that can be applied when training data becomes available gradually over time or its size is out of system memory limits.

The aim of incremental learning is to adapt to new data without forgetting its existing knowledge. Incremental algorithms are frequently applied to data streams or big data, addressing issues in data availability and resource scarcity respectively. Stock trend prediction and user profiling are some examples of data streams where new data becomes continuously available [4].

Shareable

Training Data sharing is very important so that the results can be repeated by someone else in some other environment. But, data privacy is the biggest concern. A shareable learner should be able to mask the actual data, and succinctly describe the training data so that the data is easily summarized along with preserving its properties. The data can be mutated to lower the odds of detecting the individuals in the data. The mutated data should work as well as the original data.

This is difficult to achieve because the training data can be domain specific and it's not always possible to get the similar results with the summarized data.

Context aware

Context awareness means that the miner should be able to generate different conclusions for different regions of data based on the context. Context depends on the domain of the problem that might affect the decision process. The miners should make decisions with context awareness because same data might have different meaning in different contexts and the miner should be able to take this into consideration.

In problems like recommender systems, this criteria becomes extremely important because the recommendations depend a lot on the contextual information such as time, location, or the company of other people(eg. for watching movies or dining out), thus the contextual information should be taken into account while making recommendations [18].

Self-tuning

One of the most important criteria is self-tuning of parameters of the learner. Using different values for the same parameter of the same learner algorithm can get very different results but, there can be so many different values that can be assigned to different parameters and if there are many parameters that can be changed, it gets impossible to try all the possible combinations. So, if the algorithm can itself tune its parameters, by checking most of the possible combinations, we can get the best result out of our learners.

This criterion, though very useful, is not easy to implement because a model with parameter tuning takes a very long time to execute with possible sets of parameters and find the optimal set of parameters and there are not many methods to implement a universally applicable self-tuning learner model, consuming less time.

Key Criteria

All the criteria mentioned in the above are very important but in our opinion, self-tuning is the most critical one. Parameter tuning is the process of optimizing the parameters i.e., finding the values for the algorithm parameters that give us the best possible result. The definition of best result varies from problem to problem but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters. The set of all possible solutions or values which the inputs can take, make up the search space. In this search space, lies a point or a set of points which gives the optimal solution. The aim of self-tuning is to find that point or set of points in the search space by itself.

Self-tuning is one of the most important and most useful criteria and for the implementation in this paper, we will focus on it in particular. Self-tuning is overlooked and thought to be not that useful by most of the implementations of data miners because of a misconception that the default values for the parameters are the best, which is not true.

The value of k in k-means and in KNN, the number of trees in Random forest, the depth of tree in Decision Trees and other such parameters in various learners, can result in significantly different results. Recent studies show that these classifiers may underperform due to the use of suboptimal default parameter settings. However, it is impractical to access all of the settings of the parameter space [21].

Let's talk about the impact of this criteria. There are so many problems in software engineering that can use data miners, and there are multiple learner algorithms implemented in most used languages such as Python but, there are multiple parameters in these algorithms that can have a range of values so, these miners require parameter tuning to get optimal results. If we use the best possible set of parameters for a certain problem, instead of the default parameters, the performance can improve dramatically.

Methods like grid search can be used to find the optimal set of parameters but it tests all possible combinations of parameters, which is computationally expensive. The time and computation required to find the best set of parameters, sometimes outweighs the benefits obtained by performing parameter tuning. But, it is found that evolutionary algorithms such as Differential Evolution require tens, not thousands of attempts to obtain very good results and when learning software defect predictions, this method can quickly find tunings that alter the detection precision from 0% to 60% [22]. Evolutionary algorithms, might obtain the optimal set of parameters without being computationally expensive in doing so, thus, making this criteria very useful.

2.3 Related Work

Most papers such as paper[3] and paper[28] use TFIDFvectorizer to extract the features from the data, and MultiLabel Binarizer to convert the output variable to binary matrix indicating the presence of a class label. After this, popular learners such as Naive Bayes and Linear classifiers are used with One Vs Rest Classifiers approach to perform multi-label text classification on news articles. One Vs Rest strategy consists of fitting one classifier per class. For each classifier, the class is fitted against all the other classes. This method is computationally efficient and provides easy interpretability. Since each class is represented by one classifier only, the multi-label classification problem is transformed into the combination of several single-label classifiers, making it possible to gain knowledge about the class by inspecting its corresponding classifier.

The paper[3] tries to implement these classifiers with a few combinations of parameter: alpha (laplace smoothing parameter in case of NB and regularization multiplicative term in case of SGD Classifier), parameters for TfidfVectorizer: min_df (ignore terms that appear in less than x% of documents) and max_df (ignore terms that appear in more than x% of documents) and concludes that the best result is obtained by SGD with parameter for regularization multiplicative term set to 0.001 and parameters for TfidfVectorizer set to min_df=0.001 and max_df=0.9.

But like most of the present-day miners, this solution also lacks some of the important criteria. Our key criteria, self-tuning is missing in this tool. The model might give better result if we optimize the parameters instead of using the default settings, also, it is possible that the model was working well with the current set of parameters on this particular dataset, but it won't perform the same if we used a different dataset and different domain.

2.4 Proposed Work

Parameters such as regularization term, regularization multiple, l1_ratio, maximum iterations and shuffling used for classifier algorithms, can have many possible values. We can use self parameter tuning to provide an improvement on the text-classification problem, following the similar approach as used in paper[3]. Our approach is to follow the similar procedure, we use TFIDF vectorizer and MultiLabel Binarizer for feature extraction followed by use of classifiers explained below. We will be using 2 of the self-tuning methods Random Search and Differential Evolution over these classifiers so that it can find the optimal set of parameters based on the problem at hand.

We hope to implement a better, adaptable data miner with self-tuning to get the optimized set of parameters that performs the best for any similar problem

2.4.1 Classifiers Used:

Naive Bayes: Naive Bayes is a family of algorithms based on applying Bayes theorem with a strong(naive) assumption, that every feature is independent of the others, in order to predict the category of a given sample. They are probabilistic classifiers, therefore will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be output. Naive Bayes classifiers have been successfully applied to many domains, particularly Natural Language Processing(NLP). The simple design of Naive Bayes classifiers make them very attractive for such classifiers. Moreover, they have been demonstrated to be fast, reliable and accurate in a number of applications of NLP. We have used "MultinomialNB()" from sklearn for this.

Parameters tuned:

Alpha(float) - Laplace/Lidstone smoothing parameter

fit_prior(boolean) - Whether to learn class prior probabilities or not. If false, a uniform prior is used.

Logistic Regression: Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine the outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a logit transformation of the probability of presence of the characteristic of interest. Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression), estimation in logistic regression chooses parameters that maximize the likelihood of observing the sample values[24]. We used "LogisticRegression()" from sklearn for this.

Parameters tuned:

C(float) - Inverse of regularization strength.

Tol(float) - Tolerance for stopping criteria.

fit_intercept(boolean) - Specifies if a bias should be added to the decision function.

Linear Support Vector Classifier: A supervised learning method with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. 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[25]. We used "LinearSVC()" from sklearn for this.

Parameters tuned:

C(float) - Penalty parameter of the error term.

Tol(float) - Tolerance for stopping criteria.

fit_intercept(boolean) - Whether to calculate the intercept for this model.

Linear Classifier with Gradient Descent: This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning. For best results using the default learning rate schedule, the data should have zero mean and unit variance.

This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model it fits is a linear support vector machine (SVM).

The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net). If the parameter update crosses the 0.0 value because of the regularizer, the update is truncated to 0.0 to allow for learning sparse models and achieve online feature selection[26]. For this we used "SGDClassifier" from sklearn.

Parameters tuned:

Alpha(float) - Constant that multiplies the regularization term.

l1_ratio(float) - The elastic net mixing parameter

power_t(double) - The exponent for inverse scaling learning rate.

fit_intercept(boolean) - Whether the intercept should be estimated or not.

Random Forest:

Random Forest uses an ensemble of classification trees. Random Forest uses both bagging (bootstrap aggregation), a successful approach for combining unstable learners, and random variable selection for the tree building as at each split the candidate set of variables is a random subset of the variables.

The random decision forest constructs many decision trees and each tree is grown from a different set of training data. To construct individual decision trees, training samples are randomly selected with replacement. At each splitting node, it determines its best feature from a randomly selected subspace of features. To classify a new object, each tree in the forest gives a classification and final classification of the object is determined by majority votes among the classes decided by the forest of trees. Each tree is grown fully to obtain low-bias trees and at the same time, bagging and random variable selection results in low correlation of the individual trees. The algorithm yields an ensemble that can achieve both low bias and low variance[27]. For this we used "RandomForest()" from skelarn.

Parameters tuned:

N_estimators(Integer) - Number of trees in the forest

Min_samples_leaf(float) - The minimum number of samples required to be at a leaf node. ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

min_samples_split(Integer) - The minimum number of samples required to split an internal node.

2.4.2 Tuning Methods Used:

Random Search:

Randomized search is a hyper parameter tuning technique which implements 'fit' and 'score' methods. The parameters of the estimators used to apply these methods are optimized by cross validated search over parameter settings. In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specific distributions. The number of parameter settings that are tried is given by n_iter. If all parameters are presented in the list, sampling without replacement is performed. If at least one parameter is given as a distribution, sampling with replacement is used. It is highly recommended to use continuous distributions for continuous parameters. Runtime for Randomized Search is drastically lower as compared to Grid Search. The performance is slightly worse for the Randomized Search, though this is most likely a noise effect and would not carry over to a held -out test set. RandomSearchCV() from sklearn package was used for this.

Differential Evolution:

Differential evolution (DE) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. However, metaheuristics such as DE do not guarantee an optimal solution is ever found.

DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require for the optimization problem to be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc.

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed [8].

"differential_evolution" from scipy package provided by scipy.org [14] was used to solve global optimization problems.

3 EXPERIMENTAL DESIGN

The paper[3] implements multi-label text classification on news articles data by The Guardians newspaper, to classify each article into multiple relevant categories. However, parameter tuning is not feasible on our machines on the dataset used by that paper as the data is huge and tuning is computationally challenging, so instead we decided to use two of the most widely used datasets explained below.

1. **20NewsGroups** dataset which comprises 18000 newsgroup posts on 20 topics. This is a multi-class dataset i.e. there are multiple categories and every article belongs to a single category.
2. **Reuters-21578** dataset which comprises 10788 news articles and 90 topical categories. This is a multi-label dataset i.e. every article may belong to more than one category.

We planned to use the same procedure as used by the paper[3] and implementing parameter tuning over it. We believe that parameter tuning will help to improve the performance of the classifiers we explained in previous section. This section talks in detail about our experimental setup.

3.1 Data Subsetting and Sampling

We have used multiple sampling techniques and have included all the results in the Results section. For both the datasets, we used the already available test and train sets where for "20NewsGroups" dataset, the train and test sets are divided in 60:40 and for the "Reuters-21578" dataset, the ratio is 70:30 for train and test sets.

Apart from the default train/test sets, we also performed 10 Fold cross validation to randomly divide the data into test and train sets as this approach gives better results as every data sample is considered in train and test set at least once so the results are more reliable. Because of the computation and execution time requirements, we were not able to perform 5*5 fold cross validation at this point of time, however we feel that the results by 10-fold cross validation were enough to reach our conclusions..

3.2 Setup

We chose various classifiers that could be used for this problem and performed parameter tuning for 2-3 most important parameters for every method as discussed in previous section. Apart from those classifiers we also tried with K-Nearest Neighbours and Gradient Tree Boosting classifiers but those took even longer to execute so we decided not to use them in the report.

For parameter tuning using Random Search we used "RandomSearchCV()" from the sklearn package with "n_iter" set to 100, which means 100 random combinations of the input parameters will be considered for tuning. For Differential Evolution, we used "differential_evolution()" provided by scipy.org[15] and used various settings for the parameters such as, for NP we tried 5, 10 and 20 i.e. the size of initial population will be number of parameters to tune * NP, for Crossover constant CR, we tried with 0.3, 0.9 and 0.8 and for weighing factor F we tried 0.2, 0.5 and 0.8 but we got the best results with the settings mentioned in the Practical advice section of the Differential Evolution homepage[1]. We used NP set to 10 i.e. the size of initial population will be number of parameters to tune * 10, Crossover constant CR as 0.9 and weighing factor F from the interval [0.5, 1.0].

We were able to use Differential Evolution for parameters that had Integer or Float values and use Random Search for these parameters as well as some boolean parameters too.

3.3 Evaluation Criteria

For evaluation of "20NewsGroups" dataset, we used Precision, Recall and F-Measure with micro averaging and for "Reuters-21578" dataset, we used Precision, Recall and F-Measure in both micro and macro averaging because this is a multi-label dataset.

Micro averaging: Scores computed globally over all NT × M binary decisions, where NT is the total number of test documents. In short, sum up the individual true positives, false positives, and false negatives of the system for different sets and the apply them together to get the statistics. It is a very useful test in case the dataset varies

Macro averaging: Scores computed for the binary decisions on each individual category first and then averaged over categories. It is very useful to know how the system performs across datasets.

Both micro-averaged and macro-averaged scores are informative for method comparison. The former gives the performance on each instance an equal weight in computing the average; the latter gives the performance on each category an equal weight in computing the average[23].

Apart from these, we also used stats.py[10] which performs various statistical tests to figure out whether the results obtained with self-tuning are significantly different and better than the results obtained with default parameter setting and rank various runs. We also compared the time taken by Random Search and Differential Evolution to find the optimal parameters, to determine whether self tuning is computationally feasible and whether the gains outweigh the cost we have to pay for self-tuning.

3.4 Statistical Tests

We are using several statistical tests to rank our approaches. In practice, dozens of approaches end up generating just a handful of ranks. We are using stats.py which does the following:

1. All approaches are recursively bi-clustered into ranks.
2. At each level, the approaches are split at the point where the expected values of the approaches after the split is most different to before.
3. Before recursing downwards, Bootstrap + A12 is called to check that that the two splits are actually different. If not, break.

The number of calls to the hypothesis tests are minimized by doing following:

1. Approaches are sorted by their median value.
2. Approaches are divided into two groups such that the expected value of the mean values after the split is minimized.

Hypothesis tests are called to test if the two groups are truly different. All hypothesis tests are non-parametric and include

1. Effect size tests
2. Tests for statistically significant numbers. Slow bootstraps are executed only if the faster A12 tests are passed.

As we are doing recursive bi-clustering, the hypothesis tests are called on only a logarithmic number of times. So we can say that,

1. With this method, 16 approaches can be ranked using less than ∑1,2,4,8,16 log i =15 hypothesis tests and confidence 0.99^{15} =0.86.
2. On the contrary, if we do 120 all-pairs comparisons of the 16 approaches, we would have total confidence 0.99^{120} =0.30.

4 RESULTS

4.1 RQ1: Can we get better results with self-tuning on classifiers?

We present the results in 2 parts, 1st part is for 20NewsGroups dataset and 2nd part is for Reuters-21578 dataset. Each part has a Table that presents the applicable performance measures for all 5 classifiers run with default, DEtuned (DE) and Random Search (RS) tuned parameters on default train-test split followed by a graph comparing the time taken for both DE and RS tuning for all 5 classifiers.

For 20NewsGroups dataset:

As this is not a multi label dataset, we used only micro averaged scores. Here, in the results presented in Table 1, we don’t see much happening except for Random Forest classifier where we see significant improvement in the performance measures. Results in bold show the improvement within a classifier after tuning.

Table 1. Results for 20NewsGroups

Classifier	Micro-Precision	Micro-Recall	Micro-F-Measure	Time(sec)
NB	0.81	0.81	0.81	0.4
NB-DE	**0.84**	**0.84**	**0.84**	18.74
NB-RS	0.84	0.84	0.84	59.6
SGD	0.85	0.85	0.85	1.2
SGD-DE	0.85	0.85	0.85	1480
SGD-RS	0.85	0.85	0.85	1507
SVC	0.85	0.85	0.85	4.16
SVC-DE	0.85	0.85	0.85	1200
SVC-RS	0.85	0.85	0.85	1507
LR	0.83	0.83	0.83	12.91
LR-DE	**0.85**	**0.85**	**0.85**	2010
LR-RS	0.85	0.85	0.85	1986
RF	0.65	0.65	0.65	16
RF-DE	**0.77**	**0.77**	**0.77**	4367
RF-RS	0.75	0.75	0.75	4600

In figure 1 below, we see that the time taken by both Random Search and Differential Evolution are comparable and Naive Bayes taken the least amount of time whereas, Random Forest is taking the longest time for tuning.

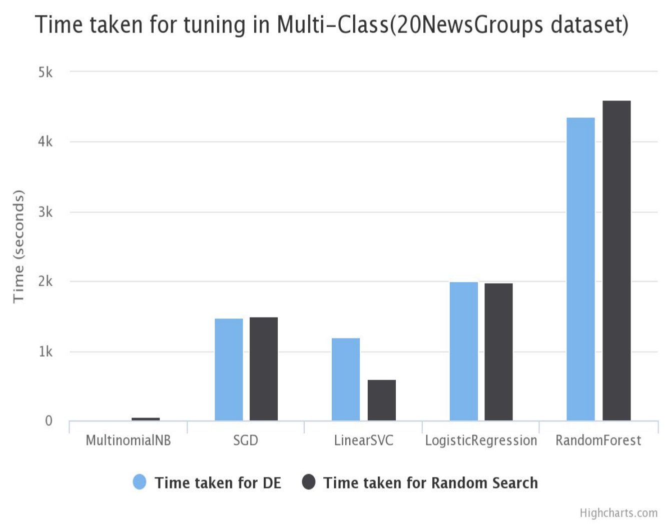


Figure 1: Comparison of time taken for tuning on 20NewsGroups

For Reuters-21578 dataset:

As this is a multi-label dataset, we have used both micro averaged and macro averaged scores. Here in results presented in Table 2, we see comparison on Micro averaged F-Measure and Macro averaged Precision, Recall and F-Measure. Clearly we see the impact of tuning on all the classifiers, where all the measures have improved a lot except for SGD classifier.
Here, we also observe that the Random Search tuning can sometime miss the optimal solution as we can see in SGD classifier where the default parameter has better performance than the Random Forest tuned. Bolded results represent the best measures for every classifier.

Table 2. Result for Reuters

Classifier	Micro F1	Macro Precision	Macro Recall	Macro F1	Time ec)
NB	0.53	0.06	0.02	0.02	1.41
NB-DE	**0.78**	**0.55**	**0.30**	**0.36**	386
NB-RS	0.78	0.51	0.26	0.33	734
SGD	0.87	0.59	0.37	0.43	2.21
SGD-DE	**0.87**	**0.60**	**0.38**	**0.45**	2190
SGD-RS	0.86	0.57	0.36	0.42	940
SVC	0.87	0.63	0.37	0.45	5.05
SVC-DE	0.87	0.66	0.43	0.50	1353
SVC-RS	**0.87**	**0.67**	**0.45**	**0.52**	1781
LR	0.75	0.27	0.1	0.13	14.74
LR-DE	**0.84**	**0.54**	**0.28**	**0.35**	2611
LR-RS	0.82	0.50	0.22	0.29	2714
RF	0.70	0.31	0.1	0.13	28.08
RF-DE	**0.77**	**0.38**	**0.15**	**0.20**	4688
RF-RS	0.75	0.38	0.14	0.18	10020

Figure 2 given below represents the time taken by tuning methods for various classifiers and again we see that the time taken by Naive Bayes is the least and that by Random Forest is the highest as it's an ensemble classifier. And the time taken by Random Search and Differential Evolution are again comparable.

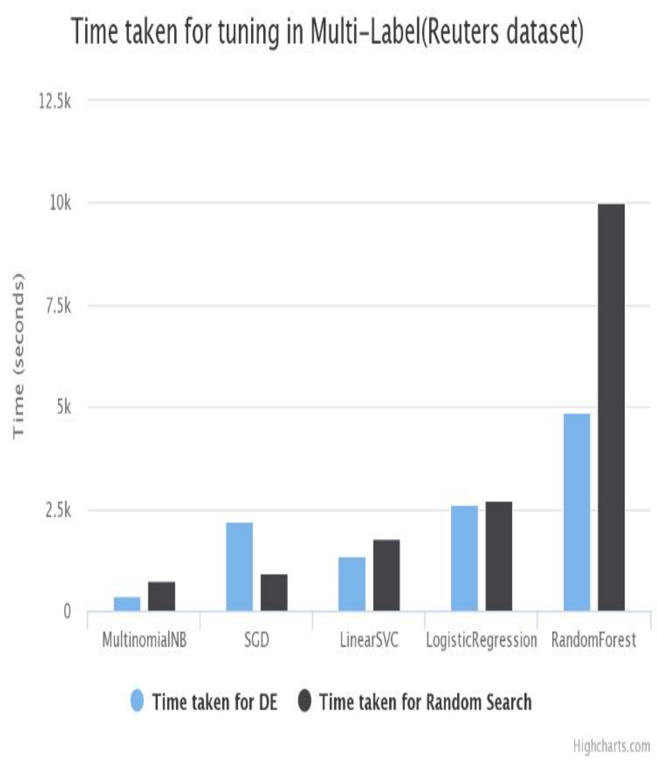


Figure 2: Comparison of time taken for tuning on Reuters

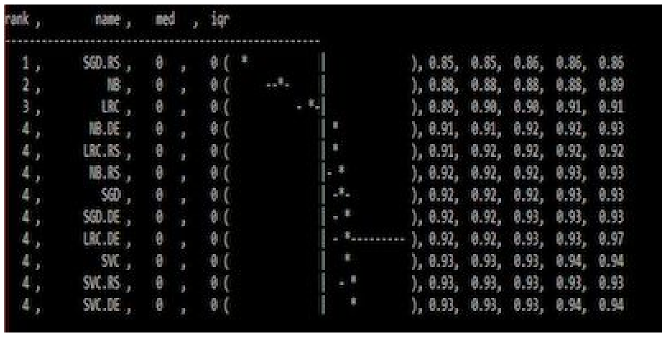
For both the datasets, we implemented both Random search and Differential Evolution for parameter tuning with various settings and on various parameters. Even though we were not able to get much success with 20NewsGroups dataset, we can confidently say that there were improvements in the performance of the classifiers after tuning and the results for Reuters-21578 dataset support this claim.

4.2 RQ2: Are the result significantly better than the solutions that ignore self-tuning?

The results of Significance tests using stats.py which is the ranking of the classifiers on 10 fold cross validation, where higher is better.

For 20NewsGroups dataset:

The findings of Table 1 follows in 10-fold cross validation too, in Figure 3 below, we can see that there is little improvement Logistic Regression and Naive Bayes and almost all the classifiers are ranked at the same level as the results are not significantly different.



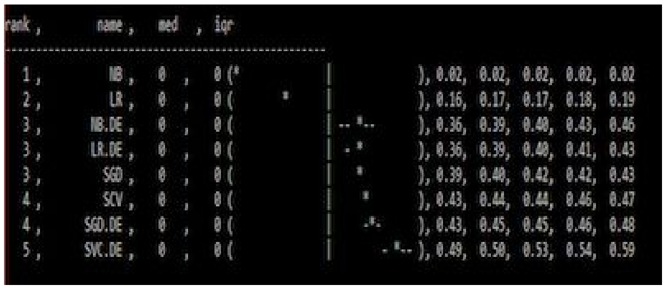
```
rank ,       name ,   med   , iqr
----------------------------------------------------
  1 ,     SGD.RS ,   0   ,   0 ( *             |           ), 0.85,  0.85,  0.86,  0.86,  0.86
  2 ,         NB ,   0   ,   0 (     --*-      |           ), 0.88,  0.88,  0.88,  0.88,  0.89
  3 ,        LRC ,   0   ,   0 (          - *-|           ), 0.89,  0.90,  0.90,  0.91,  0.91
  4 ,      NB.DE ,   0   ,   0 (               | *         ), 0.91,  0.91,  0.92,  0.92,  0.93
  4 ,     LRC.RS ,   0   ,   0 (               | *         ), 0.91,  0.92,  0.92,  0.92,  0.92
  4 ,      NB.RS ,   0   ,   0 (               |- *        ), 0.92,  0.92,  0.92,  0.93,  0.93
  4 ,        SGD ,   0   ,   0 (               | -*-       ), 0.92,  0.92,  0.92,  0.93,  0.93
  4 ,     SGD.DE ,   0   ,   0 (               | - *       ), 0.92,  0.92,  0.93,  0.93,  0.93
  4 ,     LRC.DE ,   0   ,   0 (               | - *--------), 0.92,  0.92,  0.93,  0.93,  0.97
  4 ,        SVC ,   0   ,   0 (               |   *       ), 0.93,  0.93,  0.93,  0.94,  0.94
  4 ,     SVC.RS ,   0   ,   0 (               |  - *      ), 0.93,  0.93,  0.93,  0.93,  0.93
  4 ,     SVC.DE ,   0   ,   0 (               |    *      ), 0.93,  0.93,  0.93,  0.94,  0.94
```

Figure 3: Results of Stats.py on 20NewsGroups based on F-Measure

For Reuters-21578 dataset:

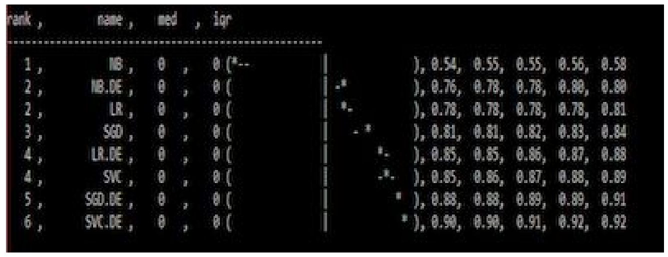
For 10-fold cross validation here, we compared only the results obtained with Default parameters and with DE tuned parameters because Random Search was producing results at most as good as the DE. We considered macro averaged F measure and micro averaged F-Measure as the objective function and Figure 4 and Figure 5 given below represent their results respectively. We also tried with optimizing considering Precision and Recall but turned out that the values for those measures after optimization were approximately equal to the results we got for those measures when optimizing on F-Measures, so we haven't included those here.

The rankings we obtained on 10-fold cross validation using stat.py again follow the result we presented in Table 2. However, unlike for 20NewsGroups dataset, here we see significant improvement in the performance. For Naive Bayes, the Macro F-Measure increased up to 40 % from 2% also the Micro F-Measure increased to high 70's from mid 50's and somewhat similar trend follows for other classifiers too. We see clear difference on rankings of most of the classifiers for both micro and macro measure, where they are being ranked higher after the tuning using DE.



```
rank ,       name ,   med   , iqr
----------------------------------------------------
  1 ,         NB ,   0   ,   0 (*              |             ), 0.02,  0.02,  0.02,  0.02,  0.02
  2 ,         LR ,   0   ,   0 (       *       |             ), 0.16,  0.17,  0.17,  0.18,  0.19
  3 ,      NB.DE ,   0   ,   0 (               | -- *--      ), 0.36,  0.39,  0.40,  0.43,  0.46
  3 ,      LR.DE ,   0   ,   0 (               | - *         ), 0.36,  0.39,  0.40,  0.41,  0.43
  3 ,        SGD ,   0   ,   0 (               |   *         ), 0.39,  0.40,  0.42,  0.42,  0.43
  4 ,        SCV ,   0   ,   0 (               |    *        ), 0.43,  0.44,  0.44,  0.46,  0.47
  4 ,     SGD.DE ,   0   ,   0 (               |     -*-     ), 0.43,  0.45,  0.45,  0.46,  0.48
  5 ,     SVC.DE ,   0   ,   0 (               |       - *-- ), 0.49,  0.50,  0.53,  0.54,  0.59
```

Figure 4: Results of Stats.py on Reuters based on Macro F-Measure



```
rank ,       name ,   med   , iqr
----------------------------------------------------
  1 ,         NB ,   0   ,   0 (*--            |             ), 0.54,  0.55,  0.55,  0.56,  0.58
  2 ,      NB.DE ,   0   ,   0 (               | -*          ), 0.76,  0.78,  0.78,  0.80,  0.80
  2 ,         LR ,   0   ,   0 (               |  *-         ), 0.78,  0.78,  0.78,  0.78,  0.81
  3 ,        SGD ,   0   ,   0 (               |    - *      ), 0.81,  0.81,  0.82,  0.83,  0.84
  4 ,      LR.DE ,   0   ,   0 (               |       *-    ), 0.85,  0.85,  0.86,  0.87,  0.88
  4 ,        SVC ,   0   ,   0 (               |      -*-    ), 0.85,  0.86,  0.87,  0.88,  0.89
  5 ,     SGD.DE ,   0   ,   0 (               |          *  ), 0.88,  0.88,  0.89,  0.89,  0.91
  6 ,     SVC.DE ,   0   ,   0 (               |           * ), 0.90,  0.90,  0.91,  0.92,  0.92
```

Figure 5: Results of Stats.py on Reuters based on Micro F-Measure

4.3 RQ3: How well do the simpler classification algorithms such as Naive Bayes and Logistic Regression perform as compared to more complex ones such as Support Vectors?

From the results presented above for RQ1 and RQ2, we can see that the performance of NB and LR improved a lot but still the Support Vector classifier performed the best for the datasets we used. However, Logistic Regression classifier wes able to match the performance of default Support Vector classifier after parameter tuning as can be seen in Figure 6 where both LR.DE (Logistic Regression tuned with DE) and SVC (Support Vector classifier with no tuning) are ranked same.

5 THREATS TO VALIDITY

Conclusions drawn from this paper should have below issues in mind.

Order and Sampling bias:

Different order and sizes of test/train samples of datasets can have an effect on the results we have obtained. We have used the default defined test and train sets for the datasets used so that someone else using the same dataset can also compare the results but there can be bias because of the data selected as test and train in default split provided with the dataset, so to mitigate this bias, we have performed 10-fold cross validation on the datasets and the statistical tests are based on the results obtained by this more robust sampling technique.

Evaluation bias:

Another important thing to consider is the evaluation criteria used to compare the performances of the classifiers with default and tuned parameters. We have used Precision, Recall and F-Measure shown in Table 1 and Table 2 for this. Though more measures can be added and reported and there might be some bias in our results, we have tried to reduce the bias associated with performance measures by using both micro averaged and macro averaged measure[28] and tuned on multiple measures and reported the results. We also tried with tuning based on Precision and Recall but the results turned out to be similar to the results we got by tuning on F-Measure so we have not included those in parameter tuning.

6 CONCLUSIONS

We produced results to show improvements in the performance of classifiers in text classification problems using self tuning approaches using both Random Search and Differential Evolution. The results were evident in case of multi-label text classification using Reuters dataset where we could clearly see the performance of most of the classifiers improved a lot with just the use of parameter tuning. However, there were no significant improvements for SGD and LinearSVC classifiers in some cases. With the results we got, we saw that there was definitely a significant amount of time required for tuning of parameters, but the improvement in measures after tuning make the cost of time and computation worth and we highly advise to use parameter tuning as most of the times there will be a definite improvement in the performance.

A logical extension of our approach is to explore more parameters that can be tuned and even try other approaches for self tuning such as genetic algorithms using DEAP[14]. We can use the current approach on some improved implementations of simple and fast Naive Bayes classifier such as Transformed Weight-Normalized Complement Naive Bayes, and Naive Bayes with locally weighted learning that really improves the results of multi-label text classification, mentioned in the paper[28] and implementing self-tuning with those approaches will definitely improve the results further.

7 REFERENCES

[1] Differential Evolution(DE): Available at: http://www1.icsi.berkeley.edu/~storn/code.html

[2] Vinod Chandola, Vipin Kumar: Anomaly Detection: A Survey. (2009). Available at: https://dl.acm.org/citation.cfm?id=1541882

[3]https://github.com/gauravsummer/ADBI-Capstone-Project/blob/master/ProjectReport.pdf

[4] Incremental Learning.: https://en.wikipedia.org/wiki/Incremental_learning

[5] Mark J. Berger: Large Scale Multi-Label Text Clasification with Semantic Word Vectors

[6] Grigorios Tsoumakas, Ioannis Katakis: Multi-Label Classification: An Overview

[7] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, Saso Dzeriski: An extensive experimental comparison of methods for multi-label learning

[8] Menzies, T. (2017) FSS17: Differential Evolution: Available at: https://txt.github.io/fss17/de.html

[9] Menzies, T. (2017) FSS17: 6 Learners(and which is “best”): Available at: https://txt.github.io/fss17/sixlearners

[10] Menzies, T. (2017) FSS17: Evaluation: Available at: https://txt.github.io/fss17/stats

[11] How to tune Algorithm Paramaters: https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/

[12] sklear.pipeline.Pipeline:: http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline

[13] Common Problems in Hyperparameter Optimization:: https://blog.sigopt.com/posts/common-problems-in-hyperparameter-optimization

[14] DEAP Documentation: http://deap.readthedocs.io/en/1.0.x/

[15] scipy.optimize.differential_evolution: https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.optimize.differential_evolution.html

[16] "Data Mining Curriculum". ACM SIGKDD. 2006-04-30. Retrieved 2014-01-27.

[17] Han, Kamber, Pei, Jaiwei, Micheline, Jian (June 9, 2011). *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann. ISBN 978-0-12-381479-1.

[18] Gediminas Adomavivius, Alexander Tuzhilin: Context-Aware Recommender Systems.

[19] Kalyanmoy Deb: Multi-Objective Optimization

[20] Michael A. Hayes, Mirian A.M. Capretz: Contextual Anomaly Detection in Big Sensor Data

[21] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, Kenichi Matsumoto: Automated Parameter Optimization of Classification Techniques for Defect Prediction Models http://chakkrit.com/assets/papers/tantithamthavorn2016icse.pdf

[22]Wei Fu, Tim Menzies, Xipeng Shen: Tuning for software analytics: Is it really necessary?

[23] Yiming Yang, Siddharth Gopal: Multilabel classification with meta-level features in a learning-to-rank framework

[24] Logistic Regression: https://www.medcalc.org/manual/logistic_regression.php

[25] Support Vector Machine: https://en.wikipedia.org/wiki/Support_vector_machine

[26] SGDClassifier: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[27] Xue-Wen Chen, Mei Liu: Prediction of protein–protein interactions using random decision forest framework- https://academic.oup.com/bioinformatics/article/21/24/4394/180192

[28] Ashraf M. Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes: Multinomial Naive Bayes for Text Categorization Revisited