PSO: An Alternative Tuning Approach

Advait Trivedi
North Carolina State University
astrived@ncsu.edu

Pranita Sharma
North Carolina State University
prsharma@ncsu.edu

Sudipto Biswas
North Carolina State University
sbiswas4@ncsu.edu

ABSTRACT

Experiment Reproducibility is a necessary factor in order to test the validity of any result. Deep learning methods are being popularly used in domains of software engineering however are not reproducible as they take several hours to train a learner. An alternative optimized SVM where the hyper parameters were tuned using Differential Evolution (DE) was suggested to drastically improve reproducibility with no degradation in quality of result. This paper uses a differently tuning approach i.e Particle Swarm Optimizer (PSO) tuned Support Vector Machine (SVM) to classify pairs of question and their answer sets to show semantic similarity within the pair. The project is motivated by result comparability on class wise performance of the deep learning model , DE optimized SVM and PSO optimized SVM and a time variant comparison of PSO and DE tuners. Statistical comparisons on precision, accuracy and F1 measure reveal that the PSO optimized SVM outperforms it counterparts on an average but not by much. DE turns out to be the faster tuner by a factor of 0.19. The findings suggest that on account of speed and performance, both tuners compared are not significantly different and can be used as substitutes.

Keywords

Particle Swarm Optimization, Support Vector Machine. Convolutional Neural Net, Differential Evolution, Parameter tuning, Search based Software Engineering.

1. INTRODUCTION

Is it always advisable to chase miniscule gains in performance even if comes at the cost of heavy computational cost? Current leanings towards deep learning as a cure all for any machine learning problem today needs to be verified before we take a leap towards it. Is it possible to replicate similar performance in lesser time? Such questions have been asked before [3]. Deep learning's utility has to be measured against the huge number of hours it takes to train even on state of the art machines running having advanced hardware and parallelization. It has been reported that even on clusters running 1000 machines (16000 cores) it still took three days to train a deep learner [38]. Deep learners also need huge datasets to be trained before they can give accurate results. Not all problems or research questions posed come with extensive datasets. This limits not only the performance but also its applicability to many problems. Another problem posed by deep learners is it basically is a black box to most users. It is hard to grasp the mathematics behind it. All these factors add to the mystery. This makes it hard to reproduce the results claimed by its proponents and check their validity. Thus even if deep learners give good results its adoption should be justified as reproducibility is a major requirement of scientific research. Can out of the box machine learning algorithms with tuned parameters give comparable or even better results? If it can the time taken to train models and give results can be reduced significantly. This is the question we have asked and answered in this paper.

In this paper we have extended the work done by by Xu et al. [4] and then by Fu, Menzies [3] to classify pairs of Stackoverflow questions along with their answers to see how semantically similar one is to the other. We have defined a set of Stackoverflow question along with its answers as a Knowledge Unit (referred hereafter as a KU). We understood after reading relevant material that it is an important problem in software engineering to know relatedness among KUs on Stackoverflow and similar forums. This if done correctly can improve the productivity of software engineers by suggesting developers related KUs to read.

Initially Xu had used a convolution neural network (CNN), a kind of deep learning method [4], to predict whether two KUs are linkable. The problem with his approach was that CNNs are computationally expensive. Even with the most state of the art hardware it still takes researchers weeks of time to train the model to get results from it. In their paper Fu and Menzies tried to improve upon the results and yet keep it reproducible. They wanted to see if a conventional machine learning algorithm but one with its parameters tuned can be used for the purpose. Fu and Menzies chose Support Vector Machine (SVM) as their learner as it had been used as a baseline in Xu's work [4]. In "Easy Over Hard"[3], Differential Evolution (DE) was used as a hyper-parameter optimizer.

To improve upon the work done previously by XU[4] and Wei [3], we chose to use a Particle Swarm Optimizer as the hyper-parameter optimizer in our technique. We also used SVM as our base learner. To compare the time taken to train another learning model with PSO we used Multi-class Logistic Regression. Our paper answers the following questions:

RQ 1: Can we reproduce learning of SVM with PSO with similar accuracy? Our results show that not only has SVM with PSO achieved similar scores across all metrics we have managed to outperform DE with SVM on a large number of metrics across all of our classes.

RQ 2: How fast does PSO tune SVM with respect to DE tuned SVM? According to our results DE with SVM is faster than PSO with SVM.

RQ 3: Can PSO replace DE for tuning a learner? No learner works best over all problems [22]. That would depend on the domain of the problem being solved. If the problem requires the learner to take less time DE would be the choice. On the other hand if the extra time taken justifies the improved results then PSO should be the optimizer of choice.

The rest of the paper is organized as Section 2 describes the background and related work , Section 3 introduces necessary background required to answer our research questions, Section 4 describes the experimental setup of our study, data sets and evaluation metrics, Section 5 presents the results, Section 6 address threats to validity, following conclusion and future works.

2. Background Material

Modern Agile approach to software engineering necessitates developers to write code to integrate with existing system and

deploy continuously so that customers can have the most current working software on their hands to use and provide feedback. Coders need to quickly find solutions to issues they're facing so that they don't hold themselves or the rest of the team back. To find answers to problems they often end up on Stack overflow and similar communities. So it becomes imperative for such communities to provide developers with relevant suggestions that can go on to answer their question. It shows that it is necessary to know what developers are currently discussing [23]. Therefore it is important to find techniques that can classify Knowledge Units (KUs) to reveal their semantic relatedness. And do so quickly.

To achieve the goals mentioned above two distinct approaches have been noticed. The first being deep learning. The second uses standard machine learning algorithms like Naive Bayes, Decision Trees, Regression, SVMs but use optimizer algorithms to tune their parameters in a way that trains it achieve better results [24]. Recently, there has been an inclination towards deep learning to solve such classification problems. But deep learning is computationally expensive. On large enough data sets it can take weeks to train [25]. Lam et al. in [26] identified localized buggy files using deep learning and that took weeks to train. It is evident that deep learning is a great drain on resources with respect to both time and money.

2.1 Deep Learning

LeCun [21] states that deep learning is a branch of representation-learning that discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. They have multiple layers of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. It is more compatible with the view that everyday reasoning involves many simultaneous analogies that each contribute plausibility to a conclusion [28]. Deep learning techniques have been proven to be successful in image classification [27], NLP and recently also in text and speech domain [21].

2.2 Parameter Tuning

Arcuri and Gordon Fraser. in their work [29] noted that in search-based software engineering (SBSE) techniques one is confronted with a multitude of different parameters that need to be chosen like population size, selection mechanism . In theory paper the authors performed the largest empirical analysis on parameter tuning in SBSE to date, collecting and statistically analysing data from more than a million experiments. These parameter tuners create a search space of parameters and moves over it in a sweep like grid search [30][32] or using heuristics [33] to search the best set of parameters. Parameters such as

- Maximum allowable tree depth in a decision tree
- Number of trees in a random forest
- Penalty parameters of logistic regression
- Types of kernels used in SVM

Agarwal et al. [10] investigated the impact of parameter tuning on Latent Dirichlet Algorithm and concluded that even though choice of parameters plays a big role in its performance only 4 out of 57 papers explored used it in any capacity.

2.3 Differential Evolution

Storn and Price [10] have explored a heuristic approach for minimizing possibly nonlinear and non-differentiable continuous space functions is presented. By means of an extensive testing it is demonstrated that the new method converges faster and with more certainty than many other acclaimed global optimization methods. The new method requires few control variables, is robust, easy to use, and lends itself very well to parallel computation. It fulfils the following major concerns of any optimization technique:

(1) Ability to handle non-differentiable, nonlinear and multimodal cost functions.

(2) Parallelizability to cope with computation intensive cost functions.

(3) Ease of use, i.e. few control variables to steer the minimization. These variables should also be robust and easy to choose.

(4) Good convergence properties, i.e. consistent convergence to the global minimum in consecutive independent trials.

2.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a optimization algorithm inspired by flocks of birds, schools of fish, or any mass movement of animals to find food while simultaneously avoiding predators and dangerous situations by employing an information sharing approach in order to develop an evolutionary advantage [36]. In PSO, there is an initial swarm of randomly generated solutions that swarms over the optimal solution over a specified number of iterations. The swarm travels based off of the large amount of information shared about the space by all the members of the swarm.

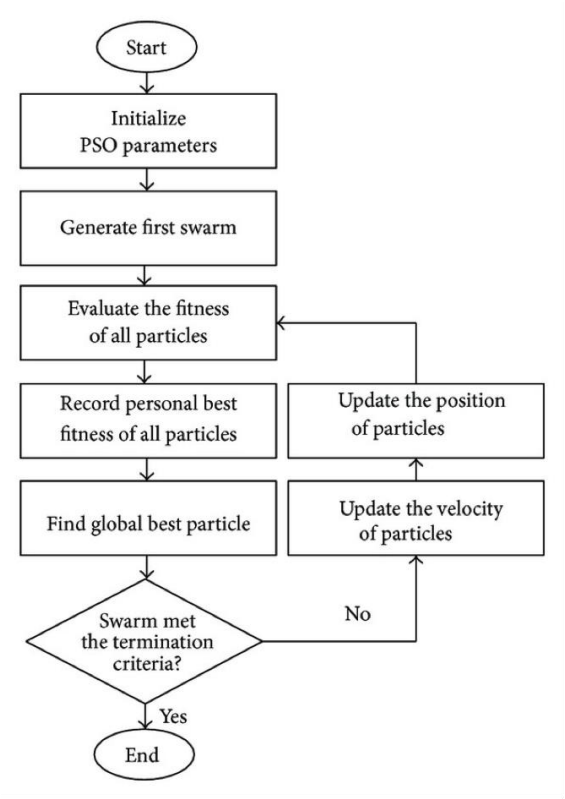
Each particle has a position and a velocity vector, and searches better position by updating the velocity vector. The velocity updating rule benefits from the particle's best position and the position of the global best particle [Eberhart, J. K. (1995). Particle swarm optimization. (pp. 1942-1946). Perth,Australia: IEEE Int. Conf. Neural Networks]. The following equations are used to update velocity and position respectively.

i) $v\,(t+1) = (w * v(t)) + (c_1 * r_1 * (p(t) - x(t)) + (c_2 * r_2 * (g(t) - x(t))$

ii) $x(t+1) = x(t) + v(t+1)$

In equations i and ii, r1 and r2 are random numbers between (0,1]. The coefficient w is the swarm inertia and is crucial to make a decision if the algorithm should rely more on its previous experience of the best position (exploitation) or try new combinations to maximize fitness (exploration). This is known as the exploration vs. exploitation trade off. The value p(t) represent the best position that particular particle has visited so far. The value g(t) represents the best position the swarm (all the particles) as a whole has realized till that time. Here c1 and c2 are local and global weights that tell the equation the extent of importance the algorithm chooses to give to either the best position visited for a certain particle (particle based exploration) or the best position captured by the entire swarm (swarm based

exploration). Flow chart 1 below shows flow of the algorithm implemented.[2]



Flow chart 1 depicts the general algorithm followed by PSO.

3. Methods

In this section we give an overview of how KUs were divided into the four categories namely, duplicate, direct link, indirect link and isolated , based on its relatedness.

For the baseline of our experiment we have chosen

1. Xu's deep learning Convolutional Neural Network with word embedding (skip-gram model in Gensim).
2. Fu an Menzies' Support Vector Machine with Differential Evolution to tune its parameters.

3.1 Learners used and Parameters to be Optimized

SVM is good at solving text classification problems. SVM seeks to minimize misclassification errors by selecting a boundary or hyperplane that leaves the maximum margin between positive and negative classes. The margin is the sum of the distances of the hyperplane from the closest point of the two classes [34].

As done previously and following Fu and Menzies' work [3] we have used the SVM module available in Scikit-learn [35], a Python package for machine learning, where the parameters shown in Table. 1 are selected for tuning. Parameter C is to set the amount of regularization, which controls the tradeoff between the errors on training data and the model complexity. A small value for C will generate a simple model with more training errors, while a large value will lead to a complicated model with fewer errors. Kernel is to introduce different nonlinearities into the SVM model by applying kernel functions on the input data. Gamma defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. coef0 is an independent parameter used in sigmod and polynomial kernel function.

Table 1 List of Tuning Parameters

Parameter	Default	Xu et al	Tuning Range
Kernel	rbf	rbf	rbf,poly,linear, sigmod
C	1	unknown	[1, 0]
Coef	0	0	[1, 0]
Gamma	1/n features	1/200	[1,0]
Random State	1		[1, 1]

The results presented below show that by exploring those above ranges, we achieved gains in the performance of our baseline methods. Even larger tuning ranges might result in greater improvements and remains a scope for future work.

4. Experimental Setup

4.1 Dataset and experimental design

Our experimental data comes from the Stack Overflow data dump of September 2016.The primary goal is to classify two questions as Duplicated, Directly linked, Indirectly linked and Isolated depending on the following definitions: [4]

Class	Description
Duplicate	These two knowledge units are addressing the same question.
Direct link	One knowledge unit can help to answer the question in the other knowledge unit.
Indirect link	One knowledge provides similar information to solve the question in the other knowledge unit, but not a direct answer.
Isolated	These two knowledge units discuss unrelated questions.

Figure 1: Depicts definition of each class of the problem[4].

In this work, we use the same training and testing knowledge unit pairs as Xu et al. and Wei et al., where 6,400 pairs of knowledge units for training and 1,600 pairs for testing. And each type of linked knowledge units accounts for 1/4 in both training and testing data.

The experiment uses pre-trained word vectors found at (https://zenodo.org/record/807727#.WirpZDdOlPY) which was granted open access by the authors of Easy over Hard experiment(Wei Fu et al.). We have adopted a similar data split strategy as our predecessor[Wei fu paper] to bifurcate the training data into new training data and tuning data, which are used during parameter tuning procedure for training SVM and evaluating candidate parameters returned by our tuner. Afterwards, the new training data is again feed into the SVM with the optimal parameters found by DE and finally the performance of the tuned SVM will be evaluated on the testing data.

To reduce the potential variance caused by how the original training data is divided, 5-fold cross-validation is performed. Specifically, each time one fold with 1280 knowledge units pairs is used as the tuning data, and the remaining folds with 5120 knowledge units are used as the new training data, then the output SVM model will be evaluated on the testing data. Therefore, all the performance scores reported below are averaged values over 5 independent runs. The experimental frame work just discussed can be seen in Figure 2. A simpler logical flow of the experimental set up can be seen in Figure 3.

Our study was also interested in comparing the variance of the tuned parameter returned by PSO. Findings could help suggest the reliance of the tuner.

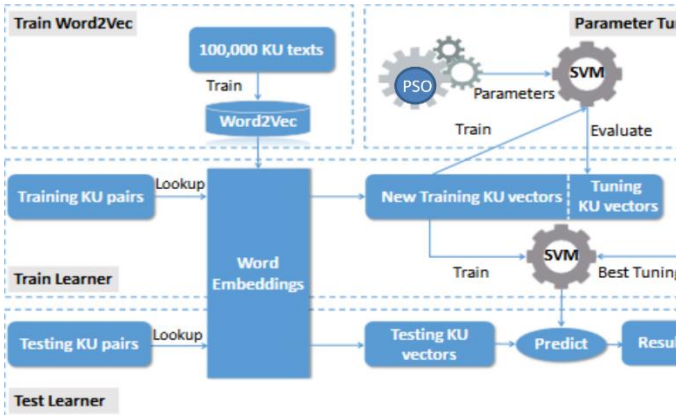


Figure 2 Depicts overview of the design experiment similar to the concept diagram of Easy Over Hard

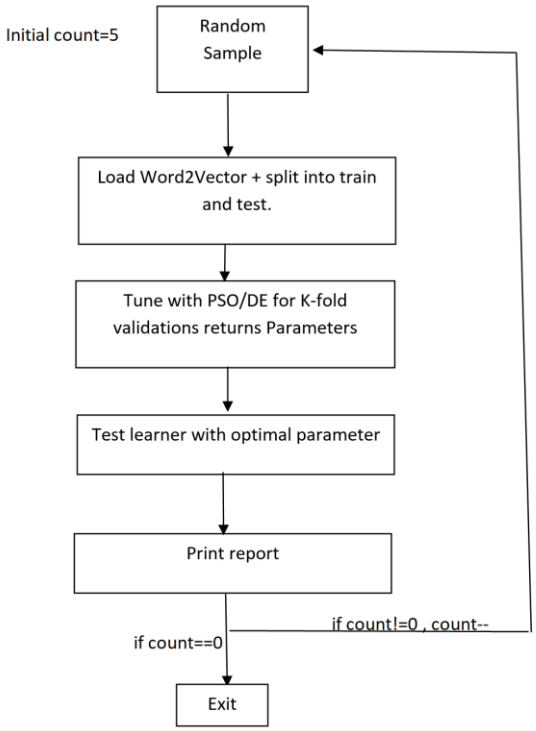


Figure 3 Exhibits logical flow of the experimental set up.

4.2 Evaluation metric

When evaluating the performance of tuning SVM on the multiclass linkable knowledge units prediction problem, consistent with XU and Wei Fu et al, we use accuracy, precision, recall and F1-score as the evaluation metrics.

Table 2 Confusion matrix With Class C_i

		Classified as			
		C1	C2	C3	C4
Actual	C1	c11	c12	c13	c14
	C2	c21	c22	c23	c24
	C3	c31	c32	c33	c34
	C4	c41	c42	c43	c44

Table 2 depicts a confusion matrix relevant to our problem. Here values along the diagonal which are represented as c_{ii} indicate the correctly classified values by our model.

Accuracy of a learner is defined as the number of correctly classified knowledge units over the total number of knowledge units. This can be represented as:

$$\text{iii) } accuracy = \frac{\Sigma_i c_{ii}}{\Sigma_i \Sigma_j c_{ij}}$$

The denominator of equation iii) i.e. $\Sigma_i \Sigma_j c_{ij}$, represents the total number of knowledge units. For a given type of knowledge units, C_j , the precision is defined as probability of knowledge units pairs correctly classified as C_j over the number of knowledge unit pairs classified as C_j and recall is defined as the percentage of all C_j knowledge unit pairs correctly classified. F1-score is the harmonic mean of recall and precision. Mathematically,

precision, recall and F1-score of the learner for class C_j can be denoted as equation iv) v) and vi) respectively:

iv) $precision_j = \frac{c_{jj}}{\Sigma_i c_{ij}}$

v) $recall_j = \frac{c_{jj}}{\Sigma_i c_{ji}}$

vi) $F1_j = \frac{2*precision_j*recall_j}{precision_j+recall_j}$

Xu and Wei have both adopted F1 score as a primary goal to be optimized for by the tuner, as it handles the trade-off between precision and recall. Taking into consideration of the work of our predecessor and popularity of the metric for classification tasks among the software community [31], [20] we too have trained our tuner to maximize F1 score.

4.3 Statistical Variability and model ranking

We have adopted to use the Scott-Knott test to report model variability and ranking. The Scott-Knott Effect Size Difference (ESD) test is a mean comparison approach that leverages a hierarchical clustering to partition the set of treatment means (e.g., means of variable importance scores, means of model performance) into statistically distinct groups with non-negligible difference.[38]. We have used stats.py from the ASE github page to run a comparative model.

5. RESULTS

In this section we present our experimental results. To answer the research questions in section 4, we conducted the following experiment –

- Compare performance of SVM tuned with DE and XU's deep learning method with SVM tuned with PSO by 5 X 5 test. It implies testing the models by 5-fold cross validation for 5 different samples of trained word2vec

The above experiment utilizes same training and test data, uses the same procedure and evaluation methods as Wei Fu [3] and XU [4], therefore we used the results reported in the work by Wei Fu [3] for performance comparison.

Table 3: Comparison of Our Method with CNN and SVM trained with DE. The best scores are marked are highlighted.

Metrics	Methods	Duplicate	Direct Link	Indirect Link	Isolated	Overall
Precision	CNN	**0.898**	0.758	0.84	0.89	0.847
	DE + SVM	0.8927	0.9072	0.9706	0.9276	0.9226
	PSO + SVM	0.878	**0.9102**	**0.982**	**0.9326**	**0.926**
Recall	CNN	0.898	**0.903**	0.773	0.793	0.842
	DE + SVM	0.9118	0.8848	**0.9882**	**0.9028**	0.922
	PSO + SVM	**0.9192**	0.8924	0.9868	0.8966	**0.9236**
F1 score	CNN	0.898	0.824	0.805	0.849	0.841
	DE + SVM	0.8968	0.8966	0.979	**0.915**	**0.9218**
	PSO + SVM	**0.8994**	**0.9014**	**0.9832**	**0.915**	**0.9248**

Table 3 demonstrates the best of the three methods for each performance metric: precision, recall and accuracy, also for each link type and overall basis.

RQ 1: Can we reproduce learning of SVM with PSO with similar accuracy?

The numeric performance quantities show that the performance of SVM tuned with PSO on an Overall basis is better than the deep learning method and DE tuned SVM. When drilling down to each metric, our method has fair performance with respect to Wei Fu's [3]. A few instances are present where deep learning numeric scores are not defeated. It is interesting to note that PSO's scores are at par with DE.

The study carried out by Mahmud Iwan [5] exhibits that DE outperforms PSO in terms of the repeatability (robustness) and the required number of iterations to bring the solution candidate into the feasible region. Furthermore, experiment performed by Rene Thomsen [6] concludes that DE outperforms PSO in the domain of numerical benchmark problems. It has an outstanding performance and it converges very fast and finds optimum in almost every run. These experiments about DE vs PSO are heavily inclined towards DE for concluding preference for tuning in optimization problems.

The revelations of our experiment do not provide an outstanding weightage to DE as the performance scores of PSO and DE are very close. SVM tuned with PSO turned out to be as good as DE tuned SVM which is a contradiction to the other studies (e.g. [5], [6]) which do not display both the tuning methods at par with each other.

How big search space did PSO seek for each of the tuning parameters to arrive at optimum result?

PSO searches for the hyperparameter values in the entire region constrained by the ranges of each of the parameters. It is vital to find if the tuner wonder the entire region to find optimum values or if it was picking up values which are the same or very close to each other. To answer this question effectively, we carried out a Variance test for the parameters.

Figure 4: Kernel vs C (penalty factor) variation

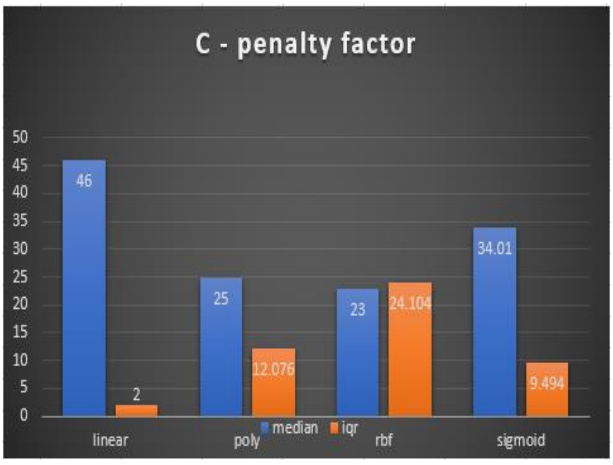


Figure 5: Kernel vs Coef0 variation

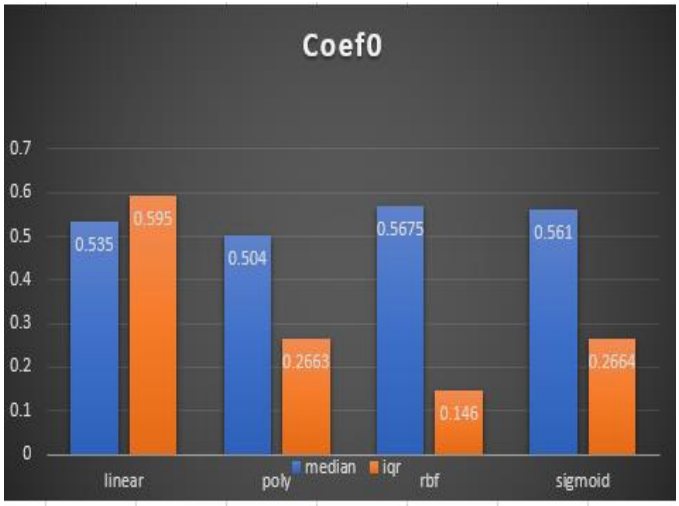
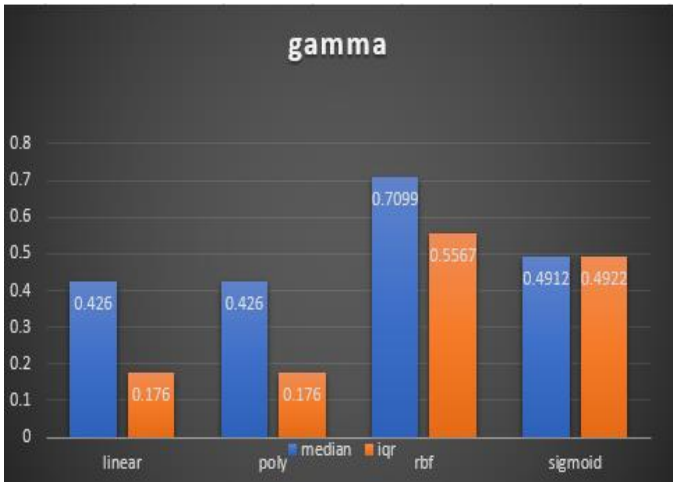


Figure 6: Kernel vs gamma variation



Figures 4,5 and 6: Median and IQR for the hyperparameters for each kernel during the tuning process for 5X5 (number of folds X number of randomly selected sample) experiment.

Median values across 5X5 experiment and IQR (a non-parametric measure of variation around the median value) are shown in these plots. They are made with the inspiration from Amritanshu Agrawal's [7] Datasets vs Parameter variation plots for LDA topic modelling.

These figures show the extent of diverse values that Coef0, gamma and C take for each kernel. Stating in terms of the search space, these depict the size of the search space which these parameters moved due to change in the particle's velocity and position according to the PSO update equations. These help to conclude whether the particle was repeatedly taken the same values or close to a particular value in each iteration or was it moving all around the search space to find the optimum value.

For example, in Figure 4., the plot for kernel = linear, portray that across the iterations, C's particles were moving in a small space around the median value, as the IQR is low. Similar trend can be seen with the plot for kernel = linear in the same figure. In contrast, referring to Figure 6., the plot for kernel = sigmoid, illustrates that gamma's particles moved around a large space to come to a final optimum value as the IQR is nearly equal to median. This kernel-wise variation helps to delineate the varied values of the hyperparameters which were considered to arrive at conclusion.

Figure 7.a: Parameter variance for entire experiment

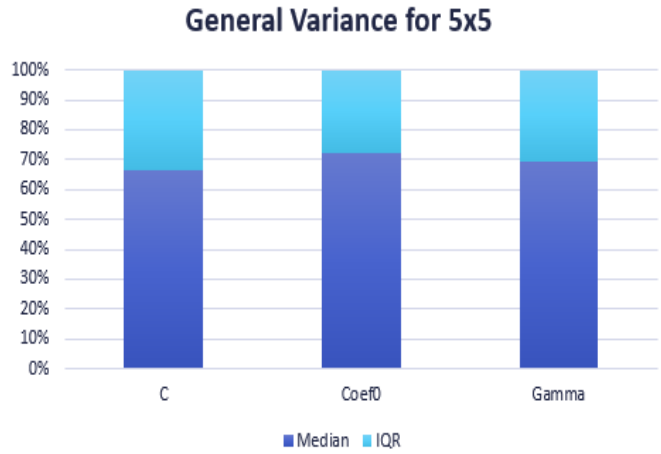


Figure 7.b: Best kernel across the experiment

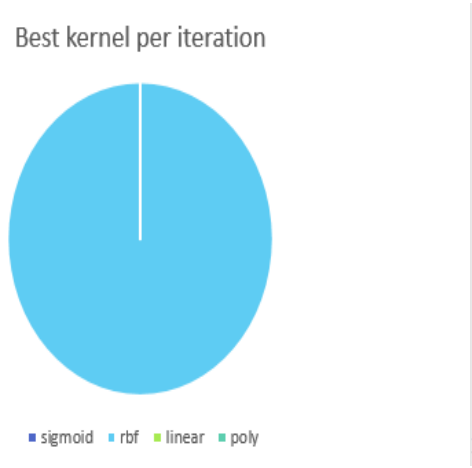


Figure 7.a. depicts overall parameter variance throughout the experiment. Also, it aids to identify the difference between default values of sklearn.svm.SVC and the actual best selected values. The default values of C, coef0 and gamma are 1.0,0.0 and 1/n respectively. According to the figure, the median of the chosen values and the default values have a considerable difference, concluding that it is vital to tune the model, before using it for training.

For instance, the default value of C (penalty factor) is 1.0 and the value selected is almost 65% higher with an IQR of 35% and the default value of Coef0 is 0.0 and the value chosen is 72% higher with IQR of 30% than the default. Therefore, for the training for a model should be carried out after tuning for best results. Iman Behravan's [8] work is on the same lines concluding that a SVM tuned with Multi objective PSO has better results than an untuned SVM in terms of performance and accuracy. Figure 7.b. shows the

best kernel selected across the iterations for tuning. It is intriguing to note that 'rbf' is selected in every single iteration as the best kernel. Also, 'rbf' is the default value for the kernel parameter set by sklearn.svm.SVC. Thus, it is the only parameter that default values got right for this experiment.

RQ 2: How fast does PSO tune SVM with respect to DE tuned SVM?

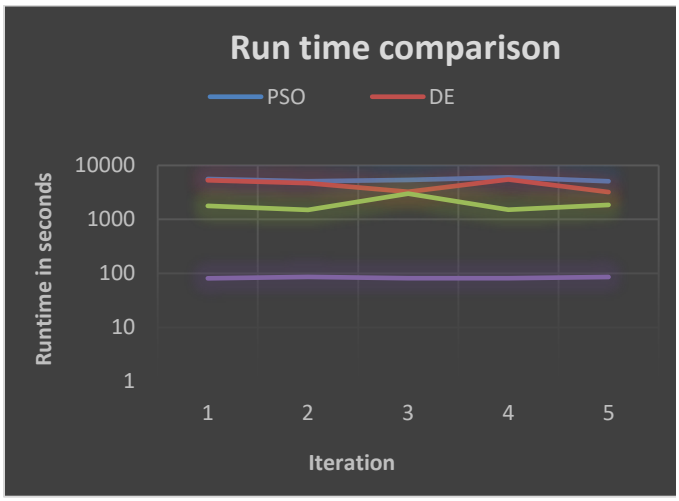


Figure 8: Run time comparison for 5X5 experiment between DE tuned SVM (orange), PSO tuned SVM (blue),PSO tuned Logistic Classifier, Untuned SVM (yellow). The y axis represents run time in seconds.

Figure 8. aids to conclude that among DE tuned SVM and PSO tuned SVM, the former is faster. Untuned SVM is very fast, which is predictable. There are few dips in the line plot for DE tuned SVM where the difference in runtime is significant. This inference is synonymous to the result of Mahmud Iwan's [5] work which states that DE converges faster than PSO while dealing with Constraint Optimization. Logistic classifier was taken into account to understand the time taken to tune a simpler learner in terms of hyper parameter search space. It can be validated that PSO Tuned Logistic learners is several magnitudes faster than any version of svm tuned.

```
f1:

rank ,          name ,    med   ,  iqr
------------------------------------------------------
   1 ,         de+svm ,    0.92  , 0.02 (             * ----------- ), 0.91,  0.91,  0.92,  0.93,  0.93
   1 ,        pso+svm ,    0.93  , 0.01 (             |  *          ), 0.92,  0.92,  0.93,  0.93,  0.93

precision:

rank ,          name ,    med   ,  iqr
------------------------------------------------------
   1 ,        pso+svm ,    0.92  , 0.01 (             *  -----      ), 0.92,  0.92,  0.92,  0.93,  0.93
   1 ,         de+svm ,    0.93  , 0.01 (--           | *---------- ), 0.91,  0.92,  0.93,  0.93,  0.93


recall

rank ,          name ,    med   ,  iqr
------------------------------------------------------
   1 ,         de+svm ,    0.92  , 0.02 (-            * ----------- ), 0.91,  0.91,  0.92,  0.93,  0.93
   1 ,        pso+svm ,    0.93  , 0.01 (      ------- | *-          ), 0.92,  0.92,  0.93,  0.93,  0.93
```

Figure 9: Scott-Knott Test ranking, the top chart talks about f1 variance, the middle one describes the precision variance while the bottom one is

The Scott-Knott test results in Figure 9 rank DE same as PSO. However it is interesting to note that for f1and precision, PSO results are a lot more stable than DE. For recall PSO exhibits some variance however the worse cases for DE are a lot worse than the worst cases for PSO. Overall PSO delivers results with lot less variance and more reliable than DE.

RQ 3: Can PSO replace DE for tuning a learner? While comparing the numeric metrics for performance comparison, PSO and DE tune SVM equivalently. On the other hand, while comparing run time, DE poses as a slight faster tuner than PSO. The overall performance metric proves PSO to be in a slightly better position than DE. The choice of PSO over DE or vice versa, therefore is entirely dependant on problem statement and the context in which slight improvement in performance metric can make a substantial difference if a little slower tuner will not debase quality of the comprehensive model. The Scott Knot results from Figure 9 reveal similar ranking of the 2, however DE exhibits more variability than PSO in the results.

6. THREATS TO VALIDITY

We should keep in mind the following when we read into the results and implications of our approach.

Dataset- For this paper we only trained and tested our model on the Stackoverflow dataset. As we have not tested it on other datasets our results may vary when our model is implemented on different datasets. Even though we have seen it to give consistently give good results across different classes of our chosen dataset we don't make any claims on its performance over other datasets. Further this experiment utilizes the pretrained word vectors used by Wei Fu et al.

Sample bias- As with any experiment the data sampled can affect the final results. Keeping this in mind we have performed a 5 fold cross validation on our data to get a uniform mix of samples. We have also run our experiments 5 times taking 5 random seeds on the pre-trained word2vec model. This essentially gives us randomized data for each of our 5 runs.

Implementation bias- We've implemented the most basic version of PSO as suggested by Eberhart et al. [9]. Since then many variations of the tuner has been shown to get different results[37][19][11]. Using these new variations can present different results from the ones we currently get. Also PSO is is highly sensitive to the selection of hyperparameter global inertia(c2) and local inertia(c1). This experiment's PSO is run uses 10 particles with w=0.75,c1=1.2 and c2=0.8. Using Different set of hyperparameters might result in different performance of PSO.

7. CONCLUSION AND FUTURE WORK RESULTS

Particle swarm optimization (PSO) [9] and differential evolution (DE) [10] have a majority of applications and particularly PSO has received increased interest from the EC community. Both techniques have shown great promise in several real-world applications [1], [12], [13], [14]. Our agenda of the experiment was to showcase an alternative approach in solving the classification problem of identifying type of links between a pair of questions. The focus was on to select a tuner devoid of complex operations such as crossover and mutation, and illustrate its power to match the performance of DE. PSO does perform with comparable metrics with respect to DE, but by taking a slightly extra time. PSO statistically is similarly ranked to PSO, however displays far lays variability than DE .Further it is worth noting that tuning a simpler learner might drastically speed up

training time on huge data-sets, thus research communities should view these factors while designing an experiment.

As Yudong Zhang's [15] survey mentions that PSO is one of the most popular methods to research upon, based on the fact that each year it has higher number of publications and articles than other algorithms such as DE, ACO et al. The forthcoming work in this domain must focus on its run time and convergence speed in comparison to evolutionary algorithms.

More tailor-made PSO algorithms such as multi-objective PSO [8], combining PSO with n-nearest-neighbour local search [16] or hybrid of PSO and DE [17] have the potential to increase the convergence speed of PSO and at the same time not compromising on the performance metrics. The upgrade in the PSO implementation can be further extrapolated to improve its search space as represented by Benoît Vallade's [18] work on PSO with dynamic search space.

8. REFERENCES

[1] Y. Fukuyama, S. Takayama, Y. Nakanishi, and H. Yoshida. A particle swarm optimization for reactive power and voltage control in electric power systems. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1523–1528. Morgan Kaufmann Publishers, 1999.*

[2] D. Gies and Y. Rahmat-Samii. Particle swarm optimization for re-configurable phase differentiated array design. *Microwave and OpticalMicrowave and Optical Technology Letters, Vol. 38, No. 3, pp. 168–175, 2003*.

[3] Wei Fu and Tim Menzies. Easy over hard: A Case Study on Deep Learning. *arXiv:1703.00133v2 [cs.SE] 24 Jun 2017*

[4] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li.2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* ACM, 51–62.

[5] Mahmud Iwan et al. Performance Comparison of Differential Evolution And Particle Swarm Optimization In Constrained Optimization. *Procedia Engineering 41 (2012) 1323 – 1328*

[6] Jakob Vesterstrom and Rene Thomsen. A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems.

[7] Amritanshu Agrawal, Wei Fu and Tim Menzies. What is Wrong with Topic Modeling? (and How to Fix it Using Search-based Software Engineering). *arXiv:1608.08176v3 [cs.SE] 8 Nov 2017.pp.12*

[8] Iman Behravan et al. An Optimal SVM with Feature Selection Using Multiobjective PSO. *Journal of Optimization Volume 2016 (2016), Article ID 6305043*

[9] J. Kennedy and R. C. Eberhart.Particle swarm optimization. *In Proceedings of the 1995 IEEE International Conference on Neural Networks, Vol. 4, pp. 1942–1948. IEEE Press, 1995*

[10] R. Storn and K. Price. Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. *Technical report, International Computer Science Institute, Berkley, 1995.*

[11] Y. Marinakis, M. Marinaki, A hybrid multi-swarm particle swarm optimizationalgorithm for the traveling salesman problem, Comput. Oper. Res. 37 (2010)432–442

[12] L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial intelligence through a simulation of evolution. In M. Maxfield, A. Callahan, and L. J. Fogel, editors, Biophysics and Cybernetic Systems: *Proc. of the 2nd Cybernetic Sciences Symposium, pp. 131–155. Spartan Books, 1965*

[13] R. Thomsen. Flexible ligand docking using differential evolution. *In Proceedings of the 2003 Congress on Evolutionary Computation, Vol. 4, pp. 2354–2361. IEEE Press, 2003*

[14] R. K. Ursem and P. Vadstrup. Parameter identification of induction motors using differential evolution. *In Proceedings of the 2003 Congress on Evolutionary Computation, Vol. 2, pp. 790–796. IEEE Press, 2003*

[15] Yudong Zhang, Shuihua Wang,and Genlin Ji. A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. *Mathematical Problems in Engineering Volume 2015 (2015), Article ID 931256*

[16] Taymaz Rahkar-Farshi et al. An improved multimodel PSO method based on electrostatic interaction using n-nearest-neighbour local search. *International Journal of Artificial Intelligence & Applications (IJAIA), Vol. 5, No. 5, September 2014.*

[17] Swagatam Das Et al. Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives. *Advances of Computational Intelligence in Industrial Systems pp 1-38*

[18] Benoît Vallade et al. Improving the Performance of Particle Swarm Optimization Algorithm with a Dynamic Search Space. *ADVCOMP 2013: The Seventh International Conference on Advanced Engineering Computing and Applications in Sciences*

[19] J.J. Liang, P.N. Suganthan, Dynamic multi-swarm particle swarm optimizer,Proc. Swarm Intell. Symp. 2005 (2005) 124–129.

[20] Sunghun Kim, E James Whitehead Jr, and Yi Zhang. 2008. Classifying so.ware changes: Clean or buggy? IEEE Transactions on So.ware Engineering 34, 2 (2008), 181–196.

[21] Yann LeCun, Yoshua Bengio, and Geo, rey Hinton. 2015. *Deep learning. Nature 521, 7553 (2015), 436–444.*

[22] David H Wolpert. 1996. The lack of a priori distinctions between learning algorithms. *Neural computation 8, 7 (1996), 1341–1390.*

[23] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering 19, 3 (2014), 619–654*

[24] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology 76 (2016), 135–146.*

[25] Wei Fu, Vivek Nair, and Tim Menzies. 2016. Why is differential evolution better than grid search for tuning defect predictors? *arXiv preprint arXiv:1609.02613(2016)*

[26] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2015. Combining deep learning with

information retrieval to localize buggy files for bug reports (n). *In Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering. IEEE, 476–481*

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classifcationcation with deep convolutional neural networks. *In Advances in Neural Information Processing Systems. 1097–1105.*

[28] Rogers, T. T. & McClelland, J. L. Semantic Cognition: A Parallel Distributed Processing Approach *(MIT Press, 2004).*

[29] Andrea Arcuri and Gordon Fraser. 2011. On parameter tuning in search based software engineering. *In International Symposium on Search Based Software Engineering. Springer, 33–47.*

[30] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classiffcation techniques for defect prediction models. *In Proceedings of the 38th International Conference on Software Engineering. ACM, 321–332*

[31] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering 33, 1 (2007).

[32] *S*tefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. 2008. Benchmarking classisscation models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering 34, 4(2008), 485–496.*

[33] James Bergstra and Yoshua Bengio. 2012. Random search for hyper- parameteroptimization. *Journal of Machine Learning Research 13, Feb (2012), 281–305*

[34] Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. *In European Conference on Machine Learning. Springer, 137–142.*

[35] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and others. 2011. *Scikit-learn: machine learning in Python. Journal of Machine Learning Research 12, Oct (2011), 2825–2830.*

[36] Panda, Sidhartha, and N. P. Padhy. "Comparison of particle swarm optimization and genetic algorithm for TCSC-based controller design." *International Journal of computer science and engineering* 1.1 (2007): 41-49

[37] T. Blackwell, J. Branke, Multi-swarm optimization in dynamic environments,Lect. Notes Comput. Sci. 3005 (2004) 489–500.

[38] Tantithamthavorn, Chakkrit, et al. "An empirical comparison of model validation techniques for defect prediction models." *IEEE Transactions on Software Engineering* 43.1 (2017): 1-18.