G9: Assessing English Language Learning using Semi-supervised Support Vector Regression

Yulin Zhao
yulin6@illinois.edu
University of Illinois Urbana-Champaign
Urbana, Illinios, USA

Haonan Sun
haonan6@illinois.edu
University of Illinois Urbana-Champaign
Urbana, Illinios, USA

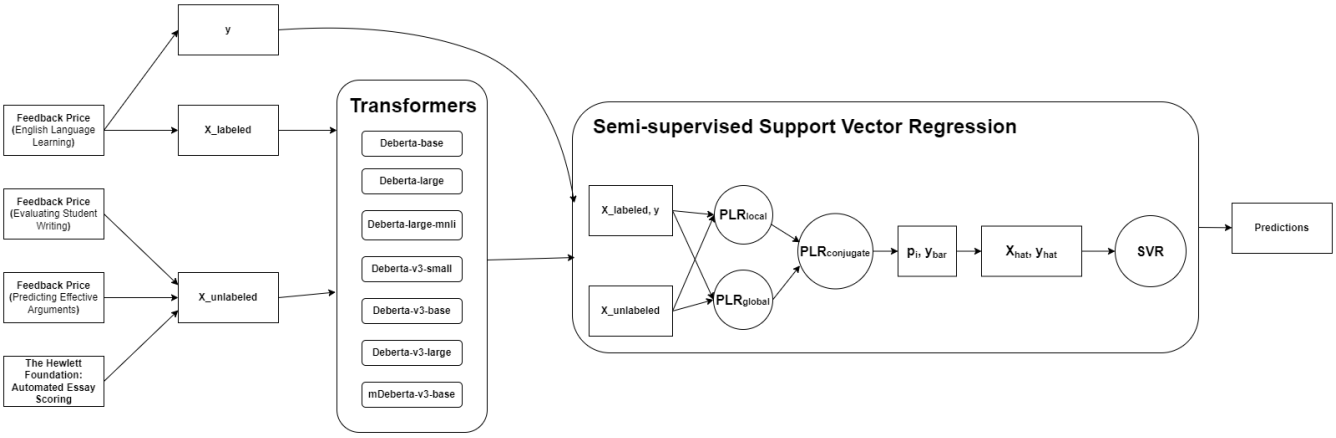

Figure 1: General Steps of the Algorithm

ABSTRACT

The assessment of English language proficiency is commonly through standardized assessments by English language educators, which is time-consuming and is based on personal or institutional criteria and experience. To standardize and automate the assessment process, we proposed a Semi-supervised Support Vector Regression (S3VR) model to provide English language assessments in six different aspects. Evaluating with 10-fold cross-validation and mean-columnwise root squared error (MCRMSE), our model achieved an average of 0.44 MCRMSE.

CCS CONCEPTS

• **Information systems → Data mining**.

KEYWORDS

Semi-supervised SVR, SVR, Co-training, Transformer, DeBERTa-V3, DeBERTa, mDeBERTa

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Yulin Zhao and Haonan Sun. 2022. G9: Assessing English Language Learning using Semi-supervised Support Vector Regression. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Nowadays, machine learning (ML) models are widely used in our daily life to automate some repetitive tasks due to their superiority of speed and accuracy. However, there are still some fields that are difficult for ML models, such as NLP tasks. For example, a recent Kaggle Competition "Feedback Prize - English Language Learning" questions whether ML models can be applied to automate the process of providing a six-dimension assessment for the language proficiency of English Language Learners. In other words, given a text, our model is expected to provide predictions on cohesion, syntax, vocabulary, phraseology, grammar, and conventions. The competition will measure the model using the mean-columnwised RMSE and time efficiency. We define this task to be a regression and NLP task. To address this question, we purposed our semi-supervised support vector regression model and achieved relatively good performance.

2 MOTIVATION

With the increasing request about the strength of the English writing capability for English learners, the current grading system is no longer effectively making reasonable and efficient judgments for language ability for ELLs. Thus, we are trying to build more efficient automated writing evaluation (AWE) systems using ML

models and NLP techniques. We believe our work will contribute to this community and help to support the integration of technology and education. Besides, our group is interested in dealing with NLP tasks and wants to learn more about semi-supervised models. In this project, we could be exposed to these fields and apply similar techniques to solve problems.

3 RELATED WORK

3.1 Datasets

The labeled dataset we used is the English Language Learning[1] provided by the Kaggle competition. It contains 3911 English essays with evaluations completed by humans, which are rated on a scale out of five on six different dimensions.

The other three unlabeled datasets are previous Evaluating Student Writing[2], Automated Essay Scoring[3] and Predicting Effective Arguments[4]also provided by the Kaggle competition. Evaluating Student Writing dataset contains 15,594 essays. Automated Essay Scoring contains 12,976 essays. Predicting Effective Arguments contains 4,191 essays. Since these three datasets have different rating metrics or different label types, these datasets will be unlabeled for our project.

3.2 NLP Transformers

Compared with other previous models, BERT (Bidirectional Encoder Representation from Transformers) [2] introduced bidirectional conversion decoding and a much larger pre-training scale. BERT creatively introduced MLM (Masked Language Model) to capture information from both left and right tokens, and has a certain probability to substitute some tokens with masks or other tokens.

DeBERTa (Decoding-enhanced BERT with disentangled attention) [5] has two main improvements, compared with former transformer RoBERTa (Robustly Optimized BERT Pretraining Approach) [8]. The first one is disentangled attention, where each word uses two vectors respectively, and the attention weights between the words are computed by using their text and relative position disentanglement matrices respectively. The second technique uses an enhanced mask decoder Electra that introduces absolute positions in the decoding layer to predict masked tokens.

DeBERTa-base and DeBERTa-large are both been proposed in the paper, the difference between these two model are the hyper-parameters select before training the model. From Fig. 2 we can directly observed that *Number of Layers*, *Hidden size* and *FNN inner hidden size* are different between two model. From Fig. 3, *Learning Rates* has different selects range during down-streaming tasks. DeBERTa-large-MNLI is the DeBERTa large model fine-tuned with MNLI task.

DeBERTa-V3 [4] has further improvements on MLM and Electra. It introduced the new disentangled attention mechanism RTD instead of MLM and an enhanced mask decoder to replace the softmax layer to solve "tug-of-war" problems.

[1]https://www.kaggle.com/competitions/feedback-prize-english-language-learning
[2]https://www.kaggle.com/competitions/feedback-prize-2021/
[3]https://www.kaggle.com/competitions/asap-aes
[4]https://www.kaggle.com/competitions/feedback-prize-effectiveness

Hyper-parameter	DeBERTa$_{1.5B}$	DeBERTa$_{large}$	DeBERTa$_{base}$	DeBERTa$_{base-ablation}$
Number of Layers	48	24	12	12
Hidden size	1536	1024	768	768
FNN inner hidden size	6144	4096	3072	3072
Attention Heads	24	16	12	12
Attention Head size	64	64	64	64
Dropout	0.1	0.1	0.1	0.1
Warmup Steps	10k	10k	10k	10k
Learning Rates	1.5e-4	2e-4	2e-4	1e-4
Batch Size	2k	2k	2k	256
Weight Decay	0.01	0.01	0.01	0.01
Max Steps	1M	1M	1M	1M
Learning Rate Decay	Linear	Linear	Linear	Linear
Adam ϵ	1e-6	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999	0.999
Gradient Clipping	1.0	1.0	1.0	1.0
Number of DGX-2 nodes	16	6	4	1
Training Time	30 days	20 days	10 days	7 days

Figure 2: DeBERTa model pretarining hyper-parameters [5]

Hyper-parameter	DeBERTa$_{1.5B}$	DeBERTa$_{large}$	DeBERTa$_{base}$
Dropout of task layer	{0,0.15,0.3}	{0,0.1,0.15}	{0,0.1,0.15}
Warmup Steps	{50,100,500,1000}	{50,100,500,1000}	{50,100,500,1000}
Learning Rates	{1e-6, 3e-6, 5e-6}	{5e-6, 8e-6, 9e-6, 1e-5}	{1.5e-5,2e-5, 3e-5, 4e-5}
Batch Size	{16,32,64}	{16,32,48,64}	{16,32,48,64}
Weight Decay	0.01	0.01	
Maximun Training Epochs	10	10	10
Learning Rate Decay	Linear	Linear	Linear
Adam ϵ	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
Gradient Clipping	1.0	1.0	1.0

Figure 3: DeBERTa model down streaming tasks hyper-parameters [5]

Hyper-parameter	DeBERTaV3$_{large}$	DeBERTaV3$_{base}$	DeBERTaV3$_{small}$	mDeBERTa$_{base}$	DeBERTaV3$_{base\text{-}analysis}$
Number of Layers	24	12	6	12	12
Hidden size	1024	768	768	768	768
FNN inner hidden size	4096	3072	3072	3072	3072
Attention Heads	12	12	12	12	12
Attention Head size	64	64		6464	64
Dropout	0.1	0.1	0.1	0.1	0.1
Warmup Steps	10k	10k	10k	10k	10k
Learning Rates	3e-4	6e-4	6e-4	6e-4	5e-4
Batch Size	8k	8k	8k	8k	2k
Weight Decay	0.01	0.01	0.01	0.01	0.01
Max Steps	500k	500k	500k	500k	125k
Learning Rate Decay	Linear	Linear	Linear	Linear	Linear
Adam ϵ	1e-6	1e-6	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9	0.9	0.9
Adam β_2	0.98	0.98	0.98	0.98	0.999
Gradient Clipping	1.0	1.0	1.0	1.0	1.0

Figure 4: DeBERTa-v3 model pretarining hyper-parameters [4]

Hyper-parameter	DeBERTaV3$_{large}$	DeBERTaV3$_{base}$	DeBERTaV3$_{small}$	mDeBERTaV3$_{base}$
Dropout of task layer	{0,0.15,0.3}	{0,0.1,0.15}	{0,0.1,0.15}	{0,0.1,0.15}
Warmup Steps	{50,100,500,1000}	{50,100,500,1000}	{50,100,500,1000}	{50,100,500,1000}
Learning Rates	{5e-6, 8e-6, 9e-6, 1e-5}	{1.5e-5,2e-5, 2.5e-5, 3e-5}	{1.5e-5,2e-5, 3e-5, 4e-5}	{1.5e-5,2e-5, 2.5e-5, 3e-5}
Batch Size	{16,32,64}	{16,32,48,64}	{16,32,48,64}	{16,32,48,64}
Weight Decay	0.01	0.01	0.01	0.01
Maximun Training Epochs	10	10	10	10
Learning Rate Decay	Linear	Linear	Linear	Linear
Adam ϵ	1e-6	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999	0.999
Gradient Clipping	1.0	1.0	1.0	1.0

Figure 5: DeBERTa-3 model down streaming tasks hyper-parameters [4]

DeBERTa-V3-base, DeBERTa-V3-large, DeBERTa-V3-small, and mDeBERTa-V3-base are all been proposed in the paper of DeBERTa-V3. From Fig. 4, the different pre-training hyper-parameters between those four models is *Number of Layers* and DeBERTa-V3-large has different *Hidden Size*, *FNN inner hidden size* and *Learning*

rates with other three models. For down-streaming tasks, those four models are different at provided different hyper-parameter for training, which are *Learning rates* and *batch size*. The main difference between mDeBERTa-v3-base and the other three models is the pre-training data. From Fig. 5, mDeBERTa-v3-base uses CC100 as the training dataset, and others use the same dataset with RoBERTa.

3.3 Support Vector Regression

SVM[3] (Support vector machine) is a very efficient algorithm that tries to find the best decision boundary while also maximizing the margin. The introduction of kernel functions allows SVM to capture nonlinear patterns and thus achieve great performance in the real world. The basic idea of SVM is to push points away from the margin boundary and enlarge the width of the margin. The optimization problem for binary classification can be written as Eq. 1.

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} w^t w + C \sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y_i - (w^t x_i + b) \geq 1 - \xi_i, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{aligned} \tag{1}$$

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} w^t w + C \sum_{i=1}^{m} \xi_i + \xi_i^* \\ \text{s.t.} \quad & y_i - (w^t x_i + b) \geq \varepsilon - \xi_i, \quad \forall i \\ & (w^t x_i + b) - y_i \geq \varepsilon - \xi_i^*, \quad \forall i \\ & \xi_i \geq 0, \xi_i^* \geq 0, \varepsilon \geq 0 \quad \forall i \end{aligned} \tag{2}$$

This simple but efficient assumption makes SVM also adapts to regression tasks. SVM for regression[10], or SVR, on the other hand, try to put points inside the margin boundary and also narrow the margin. The optimization problem can be written as Eq. 2.

3.4 Semi-supervised Learning based on SVM

Due to the great extensibility and performance of SVM algorithms, many papers are purposed to adapt SVM to semi-supervised learning (SSL) tasks. Transductive SVM (TSVM) [11] is one of the most popular methods for low-density separation, where it estimates labels for unlabeled data with max margin on both label and unlabeled data. Given that TSVM is np-hard, many studies focused on introducing new optimization methods or using graph techniques to tackle this problem, such as a Bayesian version of TSVM [1] and compact graph-based semi-supervised learning (CGSSL) [12].

Many studies focus on classification tasks, instead of regression. Among algorithms of semi-supervised regression based on SVM, COREG[13] is a co-training algorithm that adapts to self-training by using the k-NN regression model to co-train with SVR. S3VR[9], instead, modifies the optimization problem and constructs a new Lagrangian function to consider the label estimation of unlabeled data.

S3VR based on self-training [6] (S3VR-SL) is one of the most well-developed algorithms that deploys two Probabilistic Local Reconstruction [7] (PLR) models to estimate a probability distribution for each unlabeled data. Similar to the K-nearest neighbors (KNN) algorithm, PLR models find k nearest neighbors and feed them into the Eq. [3] and Eq. [4], where K refers to the kernel mapping of neighbors $K_{ij} = k(x_{NN(x,i)}, x_{NN(x,j)})$, k_* is a kernel mapping between the data point and its neighbors $k_* = [k(x, x_{NN(x,1)}), ..., k(x, x_{NN(x,k)})]$, and β is a self-defined parameter.

$$\mu_x = \sigma^2 (k^{-1} 1_{k\times 1}) \tag{3}$$

$$\sigma_x^2 = (\beta(K - 1_{k\times 1} k_*^T - k_* 1_{k\times 1}^T + k(x,x) 1_{k\times 1} 1_{k\times 1}^T) + I)^{-1} \tag{4}$$

With μ and σ^2, PLR constructs a local probability distribution of each sample. As k is the number of neighbors used for PLRs, PLR_{local} with a small k will capture local information, and PLR_{global} with a large k will capture more general information. S3VR-SL will estimate labels and choose useful data points as Eq. 6, based on the conjugated distribution as Eq. 5.

$$y_{conjugate} = \frac{\frac{y_{global}}{\sigma^2_{global}} + \frac{n y_{local}}{\sigma^2_{local}}}{\frac{1}{\sigma^2_{global}} + \frac{n}{\sigma^2_{local}}}, \sigma^2_{conjugate} = \frac{1}{\frac{1}{\sigma^2_{global}} + \frac{n}{\sigma^2_{local}}} \tag{5}$$

$$X_{hat} = X_{labeled} \cup (X_{unlabeled}, p_i \geq r), p_i = \frac{\sigma_i^2 - min(\sigma^2)}{max(\sigma^2) - min(\sigma^2)} \tag{6}$$

4 METHODOLOGY

4.1 Exploratory Data Analysis

To explore the data distribution, we plot the distribution of labels as described in Fig. 6. We can see that each label is an approximately normal distribution. Since our features are original features and long vectors after embedding, we implemented PCA to reduce dimensionality to 2 in order to plot the data distribution as shown in Fig. 7. This is highly related to what we learned from the PCA part of Chapter 3. Although there is a clear pattern of data distribution in Fig. 7, it is nearly impossible to observe any relationship with each label.

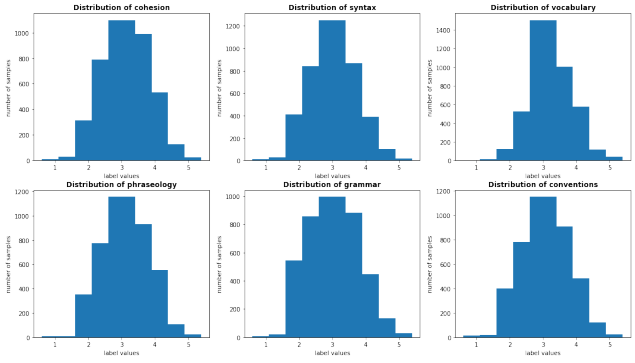

Figure 6: Distribution of each label

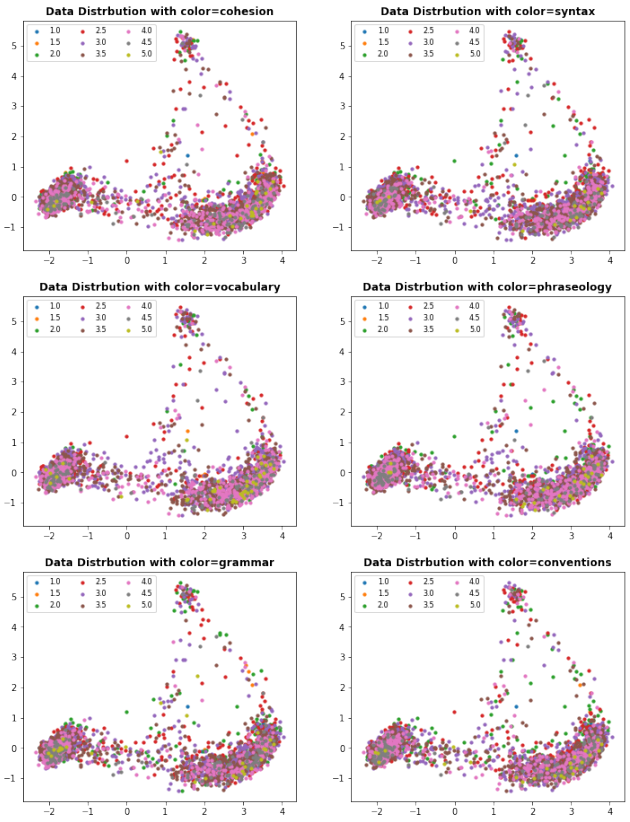

Figure 7: Data Distribution with PCA of each label

4.2 Embedding

Since our inputs are texts, we need to preprocess them into numeric features. A popular method is to introduce pre-trained NLP transformers to embed texts into vectors. NLP transformers are pre-trained deep learning models that use natural language texts as input and embed the texts into numerical vectors. This is related to deep learning part covered in Chapter 11. We chose Deberta-v3-base [4] as our baseline method due to its superiority in performance and consideration of time and computing resources. We purposed an embedding method as described in Fig. 8, where we extracted the hidden states from the pre-trained transformer and fed them into customed LSTM pooling layers.

4.3 Model

SVM and its variant SVR is widely used and researched before deep learning is highly developed due to its superiority of performance and speed. Since SVR only uses support vectors to calculate results, it is robust to outliers and performs less computation. SVR is also known as good for high-dimensional datasets or datasets with a relatively small size. Since our datasets contain 3911 samples and our embeddings has lengths ranged from 256 to 1024, we believed SVR is a good fit for our situation.

Our S3VR model is developed based on the Support Vector Regression (SVR), which evolved from the Support Vector Machine (SVM) model learned in Chapter 7, using only the labeled dataset.

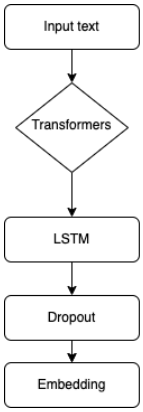

Figure 8: Embedding Architecture

It would be our baseline model for the following optimization and tests. Then we implemented the S3VR based on self-training [6] as shown in Algo 1 and Fig. 1, where it uses two PLRs [7] to estimate the labels and select the training samples. The PLR models were inspired by the KNN, which was also learned in Chapter 7, but instead were used to estimate a probability distribution. The probability distribution is then used to estimate the label value and the probability of whether it should be chosen to be used for training.

Algorithm 1 S3VR based on self learning

procedure S3VR-SL($X_{labeled}, y, X_{unlabeled}$) For
  $N_{local}(y_{local}, \sigma^2_{local}) \leftarrow PLR_{local}(X, y, k_{local})$
  $N_{global}(y_{global}, \sigma^2_{global}) \leftarrow PLR_{global}(X, y, k_{global})$
  $y_{conjugate} = \frac{\frac{y_{global}}{\sigma^2_{global}} + \frac{n y_{local}}{\sigma^2_{local}}}{\frac{1}{\sigma^2_{global}} + \frac{n}{\sigma^2_{local}}}$
  $\sigma^2_{conjugate} = \frac{1}{\frac{1}{\sigma^2_{global}} + \frac{n}{\sigma^2_{local}}}$
  $p_i = \frac{\sigma_i^2 - min(\sigma^2)}{max(\sigma^2) - min(\sigma^2)}$
  $X_{hat} = X_{labeled} \cup (X_{unlabeled}, p_i \geq r)$
  $y_{hat} = y \cup (y_{conjugate}, p_i \geq r)$
  $model \leftarrow SVR(X_{hat}, y_{hat})$
end procedure

4.4 Evaluation

Our label ranges from 1.0 to 5.0 with $step = 0.5$. To make sure each training/validation set gets a similar number of different labels, we used the multi-label stratified k-fold cross-validation to split the whole labeled data into 10 folds. In other words, we split the whole labeled dataset into ten folds, each of which contains similar fraction of different label values compared to other folds. For each training/validating iteration, it uses 9 folds as the training set and 1 fold as the validation set. To keep the same as the competition requirement, we used the mean-columnwise root of mean squared error (MCRMSE) as our metrics given as Eq. 7.

$$MCRMSE = \frac{1}{6}\sum_{i=1}^{6}\sqrt{\frac{1}{n}\sum_{j=1}^{n}(preds[j,i] - y[j,i])^2} \quad (7)$$

4.5 Optimization

4.5.1 Speed. There are two parts we have improved: estimate the probability distribution of unlabeled data and fit it into the SVR model. We translated all calculations into matrix operations and improved the speed of generating X_{hat}, y_{hat}. Then, we identified the bottleneck of our implementation: calculating the euclidean distances. After research, we surprisedly found that the distance calculation using the sklearn library is much faster than using the scipy library. Due to the time limit, we did not translate these operations into GPU operations but we could utilize GPUs to help compute the SVR using the cuML library instead of the sklearn library. So far, we believe our model is fast enough for us to train and test.

4.5.2 Performance. The length of embedding of texts is flexible and longer embeddings will contain more information. We set experiments to test the performance of using different lengths of embeddings and finally chose $length = 1024$ for our final model.

Since different transformers generate different embeddings capturing slightly different information, we introduced several transformers, including Deberta-base, Deberta-large, Deberta-large-mnli, Deberta-v3-small, Deberta-v3-base, Deberta-v3-large, and mDeberta-v3-base, as shown in Fig. 1.

We stacked the embeddings together so that there are 7168 features used in our final model. cuML implements SVR with built-in feature reduction and thus there is no need to implement extra dimensionality reduction or feature selection.

5 EXPERIMENT RESULTS

Our baseline S3VR model uses DeBERTa-v3-base to generate 256-length embeddings. We implemented multi-label stratified k-fold cross-validation with *MCRMSE* to test and validate our models. The performance of the baseline is $MCRMSE = 0.51$ with a training time of more than 20 minutes. Thus, we implemented several improvements to our model, mainly including speed optimization, trying with different settings of transformers, and hyperparameter fine-tuning.

First, we identified the speed bottleneck of our implementation and tried to optimize the speed. From Fig. 9 we can directly observe the improvement of each step. The original model takes 1961.30 seconds to train and predict. We first utilized the advantage of matrix operations and replaced most of the loops and calculations in our code, improving the running speed to 1829.99 seconds as shown in Fig. 9. We then located the bottleneck of our implementation as the function of calculating the Euclidean distance, which is used to find the nearest neighbors. We changed the distance method from the *cdist* from the scipy library to *euclidean_distance* from sklearn library, improving runtime to 1232.3 seconds again. Furthermore, we placed the SVR training/predicting process onto our GPUs using the cuML library, achieving a great improvement in runtime to 85.56 seconds. So far, we have finished optimization on speed.

The second part is to try different settings of transformers. In our baseline, we used one transformer model DeBERTA-v3-base to encode input texts into vectors. To improve our model, we tried different embedding lengths and try to use several transformers together. Changing from 256 into 1024 for the embedding length, the performance increases 0.508 to 0.485 shown in Fig. 10. Since the transformer takes a very long time (2+ hours per iteration) to encode all texts (3,000+ for labeled data and 30,000+ for unlabeled data), we did not have extra time to test the performance of more embedding lengths. Besides, different transformers generate different embeddings which may provide different information. Thus, we selected several state-of-the-art transformers (discussed in related work) and stack their output embeddings together as our features. Increased from 1 transformer to 7 transformers, the performance increases, resulting in the *MCRMSE* dropping from 0.485 to 0.448 shown in Fig. 10. These improvements definitely increase the complexity of our model or increase the runtime. As Fig. 10 shows, our S3VR takes more time to train with embeddings with 1024-length (202.08 seconds to 266.06 seconds) and with embeddings with 7×1024-length (266.06 seconds to 737.45 seconds). However, with consideration of improved performance and 737.45 seconds is still an acceptable time, we will keep using this embedding process.

The third part is to fine-tune the hyperparameters of our model. We choose the grid search method with cross-validation to find out the lowest MCRMSE score. The potetial hyperparameters are $k\_local = (10, 15, 20)$, $k\_global = (10, 20, 30)$, $\beta = (10, 100, 1000)$, $C = (0.1, 1)$. The grid search method will test each potential combination of hyperparameters. The result is shown in Fig.11. The best combination we found is $MCRMSE = 0.44$ at k_local equal to 10, k_global equal to 20, β equal to 100 and C equal to 1.

Finally, we compare the MCRMSE score between SVR and S3VR with 10-fold cross validation in Table 1. The overall score of SVR and S3VR are 0.4477 and 0.4479. It shows that our S3VR algorithm does not have obvious improvements from the baseline SVR model. However, we decide to keep using the S3VR model as our final model, since introducing 10-times-large unlabeled data may help to improve the robustness of our model.

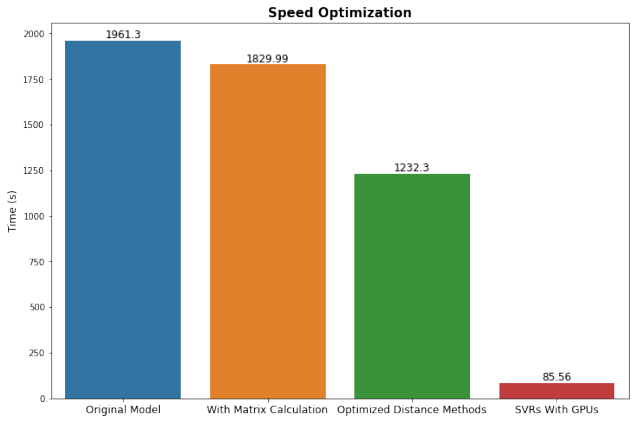

Figure 9: Speed Optimization Comparison

Table 1: Perfomance of SVR and S3VR

Experiment	1	2	3	4	5	6	7	8	9	10	*Overall*
SVR	0.4421	0.4471	0.4441	0.4485	0.4539	0.4341	0.4528	0.4334	0.4551	0.4658	0.4477
S3VR	0.4418	0.4475	0.4444	0.4491	0.4547	0.4336	0.4541	0.4329	0.4544	0.4662	0.4479

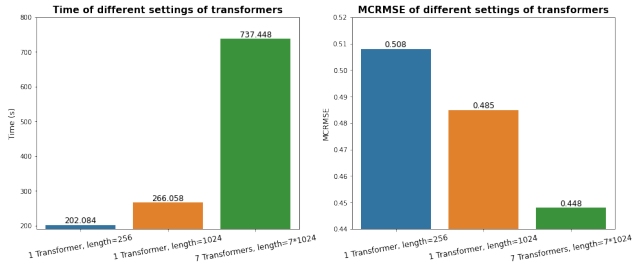

Figure 10: Time and Error Comparison of using different settings of transformers

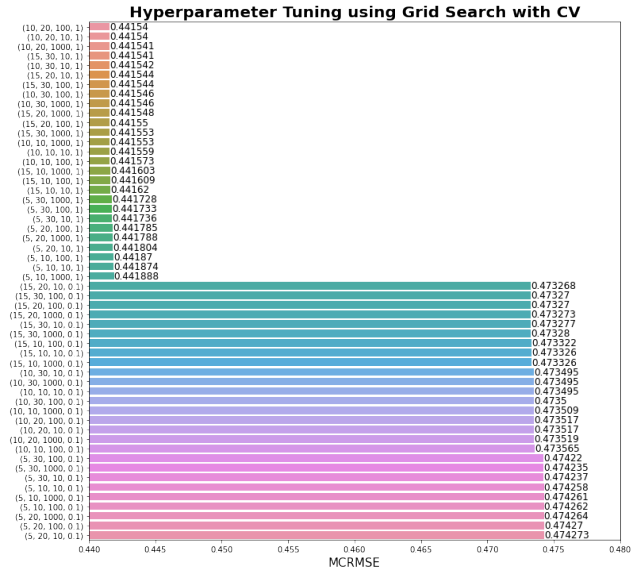

Figure 11: Error Comparison of using different hyperparameters

6 CONCLUSIONS

In this project, we proposed our solution to automate the English assessment using a semi-supervised support vector regression (S3VR) model. To convert the texts into usable numerical features, we introduced different types of NLP transformers to embed texts into vectors. We compared different embedding lengths and different numbers of transformers and decided to use the stacked features provided by 7 transformers. We implemented the semi-supervised SVR-SL proposed from [6] from scratch. We optimize its speed by simplifying calculations with matrix operations, changing distance calculation methods, and utilizing GPUs for training. We evaluated its performance using the multi-label stratified k-fold cross-validation and mean-columnwised root of squared error as metrics. Although there is little improvement from SVR to our S3VR, we believe it still increases its robustness by introducing more training data. We further fine-tuned its hyperparameters using grid search. Our final model results in an average of $MCRMSE = 0.4479$ in 10-folds cross-validation, which is very close to the scores of Kaggle top-tiers.

7 FUTURE WORK

There are several improvements that can be implemented in future work. First, the experiment results show that there is little improvement between SVR and S3VR. It might be due to the distribution of the unlabeled dataset being greatly different from the labeled dataset. We could introduce more related datasets as unlabeled datasets for training. Another potential reason is that the hyperparameter β may be not good enough. For PLR, we only test four different values for the β. In future work, we can test more values for β. Second, the most time-spending part of our model is the part calculating the PLRs, which is already transformed into a matrix operation. If these operations can be placed on GPUs, we believe the training time of our model can be further reduced. Third, in this project, due to the limitation of time and computing resources, we did not re-train the transformers. If we can freeze some layers and re-train some top layers of each transformer to fit them better on our dataset, the embedding may provide better information, and our model may result in better performance.

REFERENCES

[1] Sounak Chakraborty. 2011. Bayesian semi-supervised learning with support vector machine. *Statistical Methodology* 8, 1 (2011), 68–82.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[3] Theodoros Evgeniou and Massimiliano Pontil. 2001. Support Vector Machines: Theory and Applications. In *Machine Learning and Its Applications, Advanced Lectures (Lecture Notes in Computer Science, Vol. 2049)*, Georgios Paliouras, Vangelis Karkaletsis, and Constantine D. Spyropoulos (Eds.). Springer, 249–257. https://doi.org/10.1007/3-540-44673-7_12

[4] Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. *CoRR* abs/2111.09543 (2021). arXiv:2111.09543 https://arxiv.org/abs/2111.09543

[5] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: decoding-Enhanced Bert with Disentangled Attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=XPZIaotutsD

[6] Pilsung Kang, Dongil Kim, and Sungzoon Cho. 2016. Semi-supervised support vector regression based on self-training with label uncertainty: An application to

virtual metrology in semiconductor manufacturing. *Expert Syst. Appl.* 51 (2016), 85–106. https://doi.org/10.1016/j.eswa.2015.12.027

[7] Seung-kyung Lee, Pilsung Kang, and Sungzoon Cho. 2014. Probabilistic local reconstruction for k-NN regression and its application to virtual metrology in semiconductor manufacturing. *Neurocomputing* 131 (2014), 427–439. https://doi.org/10.1016/j.neucom.2013.10.001

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[9] Kyungha Seok. 2014. Semi-supervised regression based on support vector machine. *Journal of the Korean Data and Information Science Society* 25, 2 (2014), 447–454.

[10] Alexander J. Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Stat. Comput.* 14, 3 (2004), 199–222. https://doi.org/10.1023/B:STCO.0000035301.49549.88

[11] Vladimir Vapnik. 1998. *Statistical learning theory.* Wiley.

[12] Ming-Bo Zhao, Tommy W. S. Chow, Zhao Zhang, and Bing Li. 2015. Automatic image annotation via compact graph based semi-supervised learning. *Knowl. Based Syst.* 76 (2015), 148–165. https://doi.org/10.1016/j.knosys.2014.12.014

[13] Zhi-Hua Zhou and Ming Li. 2007. Semisupervised Regression with Cotraining-Style Algorithms. *IEEE Trans. Knowl. Data Eng.* 19, 11 (2007), 1479–1493. https://doi.org/10.1109/TKDE.2007.190644