Do Regularizer Coefficient Suits the Theoretical Prediction?

David Sriker, 203718044
Tel Aviv University
Electrical Engineering
David.Sriker@gmail.com

Matan Antebi, 300513678
Tel Aviv University
Electrical Engineering
matanantebi@mail.tau.ac.il

February 19, 2020

This project deals with the proximity between the theoretical values and the experimental values observed in real life experiments. As for the moment being although the theoretical fields are getting increased attention there is still a considerable gap between theory and reality. It is shown that for machine learning algorithms the use of regularizes improve the robustness of the algorithm with respect to generalization and stability. We will investigate how the regularize coefficient calculated in a theory prospective, which should give the best results, matches the results obtained on real and synthetically made data. The tools we are using are ones acquired in "Advanced Topics in Machine Learning" course passed by Dr. Roi Livni.

1 Introduction

The main aim in this work is to determine whether or not the theoretical calculation matches reality, second and minor objective is to check if the *Train-Validation-Test* splitting that is so commonly used is indeed necessary. Along the years there was a constant drive to match observations to theoretical results in many fields for example *physics*, *chemistry*, *engineering* etc. Those advanced in theoretical understanding have led to better performance overall so it is no wonder that it is desired to achieve it in the field of machine learning as well. Currently, the complexity of the algorithms or the lack of sufficient amount of data is limiting the theory which all it gives at the moment is some upper bounds that in some of the cases not tight at all. The second issue we are going to face is whether the validation-set is indeed necessary up until now all that deals with machine learning algorithm are separating the data to *Train-Validation-Test* while the validation-set is a subset of the train-set, which causes problem in some of the case due to the fact that in most cases we are data deficient, the reason for the splitting is quite clear but is it really necessary.
By observing the aims we mentioned above, the data we used to confirm or refute is divided into two categories: Synthesis data and common data set in machine learning, such as Mnist, Iris and Wine, which we'll dive into in 3.

All the code is available at our GitHub repository, Advanced Machine Learning.

2 Introduction terms

2.1 Machine learning

In recent years, solving a particular problem can be achieved by one of two ways. One way requires to characterize the problem explicitly and defining a tailor made algorithm to solve it. Even after characterizing the problem, not all the inputs will solve the problem and therefore the output does not converge meaningfully, means we'll not get a desired result in the system origin. On the other hand, the second way does not need requires the developer to characterize the problem explicitly. In this way the most important matter is finding a suitable model for the problem which results in desirable solution after providing many examples (of input and desirable outcomes - labels) to "learn" from.
The second method is called automated learning or in the famous name, *machine learning*.

Machine learning is an application of artificial intelligence that provides systems with the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn

from it in ways that are not trivial to humans. The input to a learning algorithm named, training data, representing experience, and the output is some expertise, which usually takes the form of another computer program that can perform some task.
We shall define some notation in the next section, but first we'll start with notation for the data-set:

$$S = \{z_1 = (x_1, y_1), \cdots, z_m = (x_m, y_m)\}$$

S is a training set of size m and $\mathcal{X}$ and $\mathcal{Y} \subset R$ are being respectively an input and an output space.

In machine learning, learning, is a wide domain topic. The learning is mostly typically classified into 2 types, although they are not the only types:

- **Supervised** algorithms can apply what has been learned in the past to new data using labeled examples to predict future events.
  Starting from the analysis of a known training data-set, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.
- **Unsupervised** algorithms are used when the information used to train is neither classified nor labeled.
  Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from data-set to describe hidden structures lies within it.

The way to assess whether our algorithm indeed performed a learning procedure on top of the data-set is to exploit the use of a *loss function,* ℓ - it's an evaluation function. It maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize an objective function, $J(\cdot)$, that in the most simplified form has the following structure:

$$J(W) = \frac{1}{m}\sum_{i=1}^{m} \ell(W, z_i)$$

While tweaking some parameters in your model the objective function will be a good indication whether you improved the model or not.
In our project we used SVM algorithm, the most commonly used loss function in SVM's is *Hinge loss* We will further discuss SVM in 2.5.

2.2 Regularization

In machine learning, regularization is the process of adding information in order to solve an ill-posed problem or to prevent over-fitting.

over-fitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.
Most classical regularization approaches are based on limiting the capacity of models, by adding a parameter norm penalty $R(\cdot)$ to the objective function. It stabilizes or limits the search space and by doing so help achieving better results in terms of generalization, which we further define in 2.3, 2.4.
There are many types of regularization, depending on the task at hand.
The common norm regularization function are:

- L_1 regularization - $\|\Sigma_{i=1}^{N} W_i\|_1$
- L_2 regularization - $\|\Sigma_{i=1}^{N} W_i\|_2$

We introduced the general form for the combination of objective function($J(W)$) and a regularization function:

$$J(W) = \frac{1}{m}\sum_{i=1}^{m} \ell(W, z_i) + \lambda R(W)$$

$\star$ The parameter λ is a tuning parameter that reflect what is the weight one want to give to the regularization term. In 2.6 we'll continue define $\lambda_{theoretical}$ and $\lambda_{optimal}$ for the main purpose - **Does the theoretical meets the optimal?**

2.3 Generalization

In machine learning, generalization usually refers to the ability of an algorithm to be effective across a range of inputs and applications. A machine learning algorithm is used to fit a model with given a certain data-set, then training the model and by meaning training the model it means to present examples from the data-set to the model. The model from epoch to epoch tweaks it's internal parameters to get better understanding the data.
Once training is over, the model is unleashed upon new data and then uses what it has learned to explain that data.
Exactly in this part raises up our second aim in the project - should the data be splitted in the classic common splitting way of *train, validation, test* or should it just be split in two categories *train and test*? The results for that sort of question is attached in section 4.
While machine learning may be able to achieve superhuman performance in a certain field, the underlying algorithm will never be effective in any other field than the one it was explicitly created for because it has no ability to generalize outside of that domain. Generalization, in that sense, refers to the abstract feature of intelligence which allows us to be effective across thousands of disciplines at once. If you over-train the model on the training data, then it will be able to identify well all the relevant information in the training data, but will fail miserably when presented with the new data. In the latter case we say that the model

is incapable of generalizing (over-fitting the training data).

2.4 Generalization & Empirical Errors

In machine learning supervised algorithms, generalization error is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data.
Because learning algorithms are evaluated on finite samples, the evaluation of a learning algorithm may be sensitive to sampling error. As a result, measurements of prediction error on the current data may not provide much information about predictive ability on new data. Generalization error can be minimized by avoiding over-fitting in the learning algorithm. The performance of a machine learning algorithm is measured using plots of the generalization error values through the learning process, which are called learning curves.
The main quantity measure that helps in assessing the performance of the trained learning algorithm is ***generalization error*** which define below [1]:

$$R(A,S) = \underset{z\sim\mathcal{D}}{\mathbb{E}_z}[\ell(A_s, z)]$$

The problem with the above expression is that it cannot be computed because the distribution $\mathcal{D}$ is unknown. Therefore, in order to compute such error we will define an estimator of the generalization error - ***empirical error*** which define below [1]:

$$R_{emp}(A,S) = \frac{1}{m}\sum_{i=1}^{m}\ell(A_s, z_i)$$

2.5 SVM

SVM, support vector machine, developed in the framework of statistical learning theory[2] [8], and have been successfully applied to a number of applications, ranging from time series prediction [5], to face recognition [7], to biological data processing for medical diagnosis [9]. The folklore view of SVM is that they find an "optimal" hyper-plane as the solution to the learning problem. The simplest formulation of SVM is the linear one, where the hyper-plane lies on the space of the input data x. In this case the hypothesis space is a subset of all hyper-planes of the form:

$$h(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x} + b;\ \ \mathbf{w}\in R^m, b\in R$$

We can show that the optimal hyper-plane, defined as the one with the maximal margin of separation between the two classes, has the lowest capacity. It can be uniquely constructed by solving a constrained quadratic optimization problem whose solution **w** has an expansion in terms of a subset of training patterns that lie on the margin. These training patterns, called support vectors, carry all relevant information about the classification problem. Our soft margin SVM written as a optimization problem has the following structure:

$$min \quad \frac{1}{2}w^T w + \lambda\sum_{i=1}^{m}\xi_i$$

$$Subject\ \ to: \xi_i \geq 0; y_i(x_i^T w + b) \geq 1 - \xi_i$$

The ***Hinge loss*** for SVM has the next form:

$$\ell_i = max(0, 1 - y_i h^\star_{x_i})$$

While $h^\star$ is the hypothesis found during the learning process of the SVM algorithm.
In our implementation we first implemented the SVM algorithm out selves, but due to the fact that we used quadratic solver, causing the running time to be dependent on the size and amount of training data, was extremely slow so eventually we used Python's library *sklearn*.

2.6 λ - *hyperparameter*

The main goal in the project was to see whether or not the theoretical value of $\lambda_{theoretical}$, from now on will be marked as $\lambda^\dagger$, will indeed matched the optimal value, $\lambda_{optimal}$, marked as $\lambda^\star$.
To do so we used the following formula for the theoretical λ:

$$\lambda^\dagger = \sqrt{\frac{8L^2B^2}{\delta m}}$$

were the parameters in the formula stands for:

- m - is the total number of samples.
- L - is the Lipschitz of the function, it was taken to be 1 because we used Hinge Loss and it is *1-Lipschitz*.
- B - is the bound of the group, it was taken to be the max value of the data.
- δ - is the sampling probability, it was taken to be 1 because we used the expectation measure.

Therefore, at the end we get:

$$\lambda^\dagger = \sqrt{\frac{8B^2}{m}} \propto \sqrt{\frac{1}{m}}$$

The formula was taken from [6].

3 Data

In this section we'll describe the different types of data sets (3.1, 3.2, 3.3, 3.4) that our SVM model has been trained. We used several types of data-sets so we could get into much brighter conclusion about the relationship between $\lambda_{theoretical}$ and $\lambda_{optimal}$ (later on those λs will be defined differently).

3.1 Synthetic data

we created synthetic data-set base on multivariate normal distributions, we focused on the 2-dimensional space.
Let define the jointly probability density function (pdf) of two random variables x, y were:

$$\mu = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \Lambda = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma y \\ \rho\sigma_y\sigma x & \sigma_y^2 \end{bmatrix}$$

μ and λ stands for expectation and covariance matrix respectively, and for the data-set we created $\rho = 0$, therefore the jointly pdf of two independent random variables is given below:

$$f_{XY}(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}[\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2}]}$$

We have defined two Gaussians with the following expectations and covariance's matrices:

$$\mu_1 = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix} \Lambda_1 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.6 \end{bmatrix}$$

$$\mu_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \Lambda_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.2 \end{bmatrix}$$

It is important to note that the synthetic data-set isn't linearly separable (more info is attached in 4), as can be seen in figure 1.

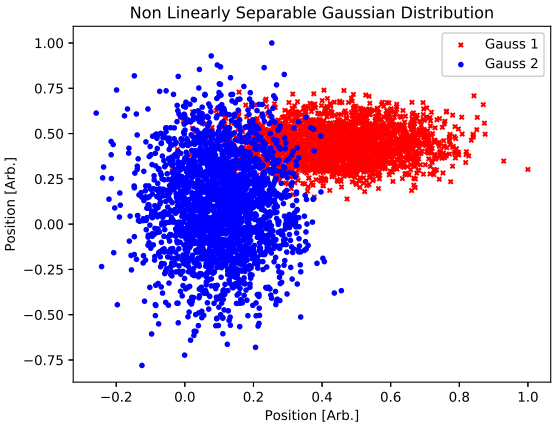

Figure 1: *Non Linearly Separable Gaussian's*

In addition to the synthetic data, we implemented real data-sets such as Iris, Wine and Mnist. We have performed the same processes for the IRIS, Wine & Mnist data-sets as we did for the synthetic data.

3.2 Iris

The Iris data-set [3] is a common starting point to most who begin in the field of data science or machine learning. Although it is the first starting point is is often a common pitfalls and one should be careful in interpreting the results.
The data-set contains 3 classes of 50 instances each, where each class refers to a type of iris plant: *Setosa, virginica & versicolor*.
One class is linearly separable from the other 2 and the latter aren't linearly separable from each other.

3.3 Wine

We examined the Wine data set from [3], this data set contain the results of the chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The constituents are:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

3.4 Mnist

The Mnist data-set of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image. This technique of resetting the average and normalizing the variance is important for making the training model to converge faster than training model without using this technique.
The reason for training the model with this data-set was due to that we wanted to focus our main efforts into the project's goals and not waste time and resources for preprocessing complicated data. The data is available for downloading in many ways and sources, one of them attached here [4].
Sample of the Mnist data-set can be seen in figure 2.

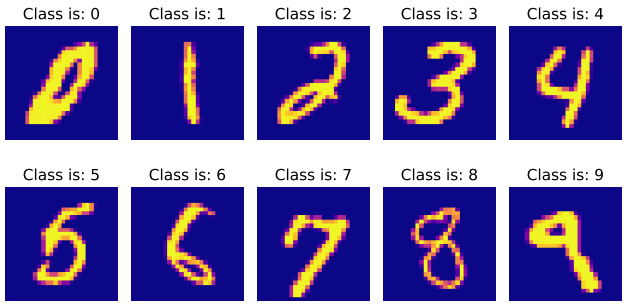

Figure 2: *Mnist Digits*

4 Results

This section presents the results for the λ's that was tested during the SVM learning for all kinds off data-sets (3)
In order to examine the above we used the soft margin SVM algorithm [2], where we used a standard linear kernel. The solver is based on quadratic programming, QP, thus the run time depends on the number of samples, in the cases where the data isn't contain only 2 classes we took in each run 2 clsses and binarized their labels to be $[-1, 1]$ and applied the soft-SVM algorithm.
In the next subsection (for each one of the data-sets), the results will be presented in tables for the λ's (*theoretical and optimal* values for the *validation accuracy* and *test accuracy* result in $[\%]$. In addition we added some scatter figures which describes the accuracy Vs. λ's values for both the Validation and Test and we highlight the score that was the highest result in each case.

4.1 Synthesize data-set

As mentioned earlier we synthesized jointly Gaussian pdf data-set. The data-set isn't linearly separable and the reason for this choice is the fact that if the data was linearly separable then we would have got $100[\%]$ accuracy and the analysis of the regularizer effect would have been difficult.
In figure 3 the results presented are for the accuracy on both the validation and test sets.

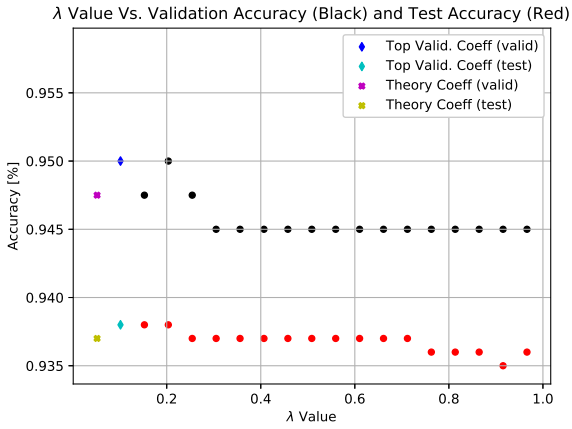

Figure 3: *Gaussian's Accuracy on Validation/Test Sets*

The results are summarised in table 1. From the results we can see that the theoretical values for $\lambda^{\dagger}$ is quiet similar for the $\lambda^{\star}$ values obtained from the validation, we observe that $\lambda^{\dagger}$ is different from the $\lambda^{\star}$ for the best test case while although different from the best performance on the test the results are within reasonable range of error. we can conclude that the theoretical value is merged with practical observations.

Table 1: *Gaussian's Case Results*

	Value	**Validation Accuracy**	**Test Accuracy**
$\lambda^{\dagger}$	0.05263	95.2[%]	93.7[%]
$\lambda^{\star}$ **(valid)**	0.05085	95.2[%]	93.7[%]
$\lambda^{\star}$ **(test)**	0.10169	95.2[%]	93.8[%]

4.1.1 Iris

With Iris data-set which has 3 - classes (0,1 and 2) we took all the permutation and stored it in table 2. We'll focus on the permutation which include class 0, in those permutations we may see that we get $100[\%]$ accuracy this is due to the fact that this class is linearly separable and we cannot conclude regarding the λ's values nor about the splitting into *Train-Validation-Test*. For the (1,2) case we see that although the $\lambda^{\dagger}$ is different from the $\lambda^{\star}$ for the best test case we get the same results which indicate that taking the $\lambda^{\dagger}$ as the theory suggest would have given us satisfying results. Regarding the *Train-Validation-Test* splitting we can see that indeed the Test results is identical which is an indication that the splitting is indeed meaningful. In figure 4 the λ's results are attached for classes (1,2).

Table 2: *Iris Results*

IRIS RESULTS				
	Classes	**Value**	**Val Acc.**	**Test Acc.**
$\lambda^{\dagger}$	**[0,2]**	0.46882	100%	100%
$\lambda^{\star}$ **(Valid)**		0.10169	100%	100%
$\lambda^{\star}$ **(Test)**		0.10169	100%	100%
$\lambda^{\dagger}$	**[0,1]**	0.38551	100%	100%
$\lambda^{\star}$ **(Valid)**		0.25423	100%	100%
$\lambda^{\star}$ **(Test)**		0.25423	100%	100%
$\lambda^{\dagger}$	**[1,2]**	0.46568	62.5%	85%
$\lambda^{\star}$ **(Valid)**		0.71186	87.5%	85%
$\lambda^{\star}$ **(Test)**		0.05084	62.5%	85%

4.1.2 Wine

Wine data-set also has 3 classes (0,1 and 2) and we again included all the permutations in table 3. We can see that in the permutations including class 1 the *Train-Validation-Test* splitting is justified, while the Test results is a bit far from the results obtained for the $\lambda^{\dagger}$ in the case we would have taken it the results stays almost the same. In the (0,2) permutation we see that both the *Train-Validation-Test* splitting and the $\lambda^{\dagger}$ results performed poorly on the Test, in Figure 5 the λ's results are attached for classes (0,2).

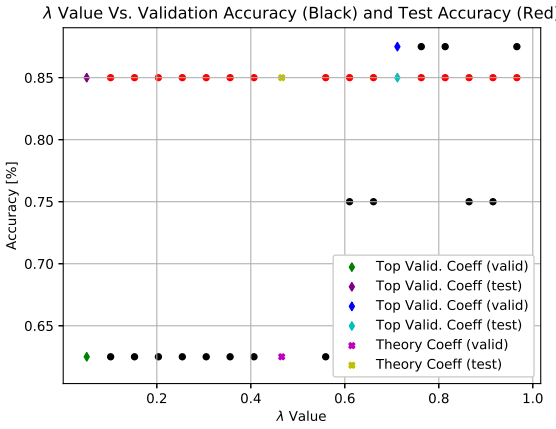

Figure 4: *Iris Accuracy on Validation/Test Sets [classes* 1,2*]*

Table 3: *Wine Results*

WINE RESULTS				
	Classes	**Value**	**Val Acc.**	**Test Acc.**
$\lambda^{\dagger}$	**[0,2]**	0.29748	87.5%	63.6%
$\lambda^{\star}$ **(Valid)**		0.30508	87.5%	72.7%
$\lambda^{\star}$ **(Test)**		0.45762	87.5%	86.3%
$\lambda^{\dagger}$	**[0,1]**	0.29236	100%	84.6%
$\lambda^{\star}$ **(Valid)**		0.30508	100%	84.6%
$\lambda^{\star}$ **(Test)**		0.15254	90%	88.4%
$\lambda^{\dagger}$	**[1,2]**	0.17268	44.4%	62.5%
$\lambda^{\star}$ **(Valid)**		0.05084	44.4%	62.5%
$\lambda^{\star}$ **(Test)**		0.05084	44.4%	62.5%

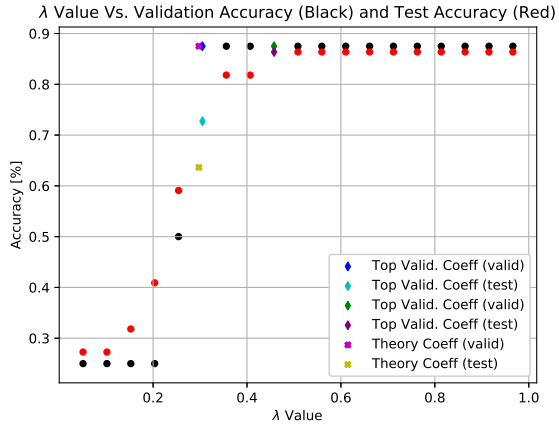

Figure 5: *Wine Accuracy on Validation/Test Sets [classes* 0,2*]*

4.1.3 Mnist

Now we didn't checked all the permutations, because then we would need 32 table entries which will overwhelm, we checked 6 options instead which we believe to convey our hypothesis. By just glancing to table 4 we can see that *Train-Validation-Test* splitting is indeed performing well and it indeed reflect on the Test case. We see that although the $\lambda^{\dagger}$ is different from the $\lambda^{\star}$ for the best test case the results are in the tolerance range and we can easily conclude that taking the theoretical value is sufficient. We then plotted the accuracy of the validation-set as well as the test-set as can be seen in figures 6, 7 for 2 cases respectively:

- digits [1,8]
- digits [2,9]

Table 4: *Mnist table results*

MNIST RESULTS				
	DIGITS	**Value**	**Val Acc.**	**Test Acc.**
$\lambda^{\dagger}$	**[1,8]**	0.4800	98.2%	98.4%
$\lambda^{\star}$ **(Valid)**		0.08474	98.3%	98.5%
$\lambda^{\star}$ **(Test)**		0.08474	98.3%	98.5%
$\lambda^{\dagger}$	**[2,9]**	0.48020	98.9%	98.4%
$\lambda^{\star}$ **(Valid)**		0.16949	99.2%	98.6%
$\lambda^{\star}$ **(Test)**		0.08474	98.9%	98.7%
$\lambda^{\dagger}$	**[3,7]**	0.44899	98.1%	98.1%
$\lambda^{\star}$ **(Valid)**		0.08475	98.6%	98.3%
$\lambda^{\star}$ **(Test)**		0.08474	98.6%	98.3%
$\lambda^{\dagger}$	**[3,8]**	0.49207	97.1%	95.8%
$\lambda^{\star}$ **(Valid)**		0.08474	97.4%	96.1%
$\lambda^{\star}$ **(Test)**		0.08474	97.4%	96.1%
$\lambda^{\dagger}$	**[6,7]**	0.45629	100%	99.8%
$\lambda^{\star}$ **(Valid)**		0.15254	100%	99.8%
$\lambda^{\star}$ **(Test)**		0.05084	99.8%	99.8%
$\lambda^{\dagger}$	**[3,9]**	0.45496	98.5%	98.1%
$\lambda^{\star}$ **(Valid)**		0.05084	98.7%	98.3%
$\lambda^{\star}$ **(Test)**		0.05084	98.7%	98.3%

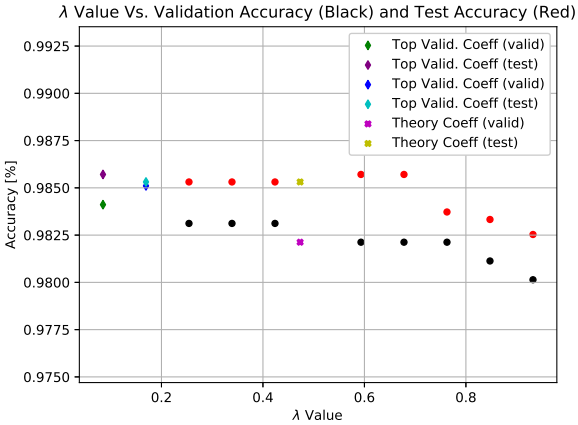

Figure 6: *Mnist Accuracy on Validation/Test Sets [digits* 1,8*]*

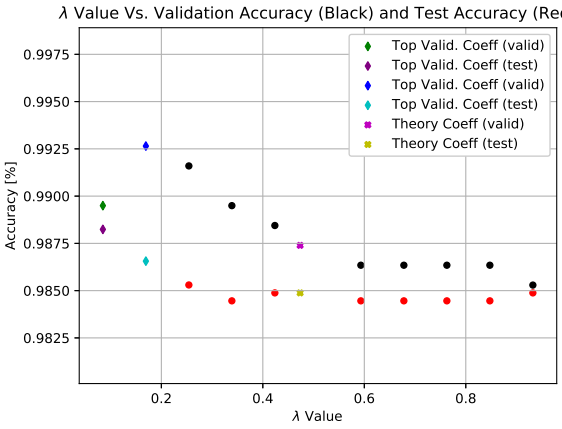

Figure 7: *Mnist Accuracy on Validation/Test Sets [digits 2,9]*

5 Conclusions

In the previous section, 4, we discussed the results we got for all data-sets that we introduced in section 3. Based on the results we got we are able to answer the two tasks we wished to explore in this work:

1. Does the theoretical value for the λ matche reality?
2. Does the common splitting *Train-Validation-Test* is indeed necessary?

First, for the theoretical λ verses the optimal one we see that the results are indeed show that the theoretical value, which may be different as shown in the results, is often gives results which are in the tolerance range and indeed the theory is in this sense coincide with reality. The difference exist mostly because the real data-sets are not "behaved" as the theory assume and that fact causes deviations, while in the synthesized data-set those assumptions are met and we indeed get conclusive answer.
Second, for the common division *Train-Validation-Test*, as we see it, the division is indeed relevant and capture the general distribution that represent the Test data and in the majority of cases the values for the λ is identical.

In conclusion, we did indeed get results that indicate that λ's theoretical value converges to a practical value for validation and test, but its value itself did not always come close to the value measured in practice. Regarding the common division (Train-Validation-Test) we are pretty sure that this division is necessary and because we did not get a contradiction with the results regarding the division we saw no reason to doubt this splitting.

Future work that can strengthen our hypothesis can be in 2 directions that can complement on each other. First, we suggest examining the multi-class case, where there it will be even harder to find a linear hyper plane that will separate the classes well and then the value of λ will have a significant role that might show whether the theory can than match reality. Second, try to perform the same task with a small neural network where we will use weight regularization which will have 2 advantages with respect to the SVM algorithm. The first is that we will be able to use the *Train-Validation-Test* splitting and adjust the training with respect to the Validation and see how it affect the results; the second is that we will see whether we can find correlation between the λ value the theory predict for the linear case and data that "behaves" well between the actual results we will observe.

References

[1] Olivier Bousquet and André Elisseeff. "Stability and Generalization". In: *J. Mach. Learn. Res.* 2 (Mar. 2002), pp. 499–526. ISSN: 1532-4435. DOI: 10.1162/153244302760200704. URL: https://doi.org/10.1162/153244302760200704.

[2] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Mach. Learn.* 20.3 (Sept. 1995), pp. 273–297. ISSN: 0885-6125. DOI: 10.1023/A:1022627411411. URL: https://doi.org/10.1023/A:1022627411411.

[3] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[4] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[5] Junshui Ma, James Theiler, and Simon Perkins. "Accurate On-line Support Vector Regression". In: *Neural Computation* 15.11 (Nov. 2003), pp. 2683–2703. DOI: 10.1162/089976603322385117. URL: http://dx.doi.org/10.1162/089976603322385117.

[6] Shai Shalev-Shwartz et al. "Stochastic Convex Optimization". In: *COLT*. 2009.

[7] A. Tefas, C. Kotropoulos, and I. Pitas. "Using support vector machines to enhance the performance of elastic graph matching for frontal face authentication". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.7 (July 2001), pp. 735–746. DOI: 10.1109/34.935847. URL: http://dx.doi.org/10.1109/34.935847.

[8] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[9] K. Veropoulos, N. Cristianini, and C. Campbell. *The Application of Support Vector Machines to Medical Decision Support: A Case Study*. 1999.