Computational Intelligence 2 - Second Homework

Gabriel S. Vieira
PESC/COPPE - UFRJ
Federal University of Rio de Janeiro
Rio de janeiro, Brazil
gabrielv@cos.ufrj.br

***Abstract*—This paper is a description of the second homework of Computational Intelligence 2, a course from the Systems Engineering and Computer Science Program(PESC) associated with COPPE-UFRJ. This homework utilizes Support Vector Machines(SVM) to classify the Banana dataset and check its performance with different hyperparameters. Also, we use Convolutional Neural Networks(CNN) to classify handwritten digits. We utilize the MNIST dataset for this method of classification.**

***Index Terms*—SVM, Banana Dataset, MNIST, CNN, Machine Learning**

I. Introduction

In our first experiment, we deal with the Banana Dataset, an artificial data set where instances belong to several clusters with a banana shape [1]. Our task is to correctly classify which point on a Cartesian space belongs to one banana shape or the other. Also, we need to mention that this data set is not linearly separable. For this task we are going to use SVMs as a classifier, it is an advanced technique that works well on non-linearly separable data. We are going to use a specific hyperparameter called kernel.

Furthermore, our second experiment is based on handwritten digits images classification. In this experiment, we used a dataset with handwritten digits images called MNIST [2]. To classify those images, we chose Convolutional Neural Networks as our classifier. CNNs are good classifiers for images since they preserve image relations while they reduce their dimensionality, creating a powerful neural network using less information.

II. Support Vector Machine Experiment

A. Experiment details

In our experiment we are asked to use SVM in order to classify the banana dataset. This dataset has 2 classes (-1) and (+1) and has 5,300 observations with 2 attributes (x) and (y) , representing the observation position on Cartesian space.

We also need to check different hyperparameters performance on SVM as asked, we need to choose these kernels: linear, polynomial, RBF, and sigmoid. In this last kernel, the hyperparameter varies between 1, 0.05, and 0.01.

To have a good estimate of our results, we are going to use cross-validation using k-fold on our experiment, but we are asked to use different values for k. More explicitly 2, 5, and 10 folds to show the performance on each one.

After seeing SVM's performance we need to choose a different classifier for this task and compare the best SVM result with this new classifier. For this classifier, we are choosing K-Nearest Neighbors as a classifier(KNN) [4].

B. Support Vector Machines

SVM is a technique for two-group classification problems. The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high dimensional feature space. In this feature space, a linear decision surface is constructed. Special properties of the decision surface ensure a high generalization ability of the learning machine [3].

To generate such a decision surface, we utilize kernel methods to map our data into higher dimensional spaces in which our data are probably linear separable. Many different kernel functions can be utilized on this kind of transformation, but the most common are: Linear, Polynomial, Sigmoid, and RBF. These kernel functions can be seen on equations 1 ,2 ,3 ,4 respectively.

$$k(x, y) = x^T y \tag{1}$$

$$k(x, y) = (\gamma x^T y + r)^d \tag{2}$$

$$k(x, y) = \tanh(\gamma x^T y + r) \tag{3}$$

$$k(x, y) = \exp(-\gamma ||x - y||^2) \tag{4}$$

Only the linear kernel function is not suitable for the last kernel description. It does not map our data into higher dimensional space, but instead, it tries to generate our decision surface on our current space. If we have linearly separable data we can have a Hard-margin using a linear kernel to separate data with maximum margin, otherwise, we should have a Soft-margin that allows miss-classification to ensure good generalization.

C. Results

Our first task is to plot our dataset and see it. We can check a scatter plot of each data with its classification in Figure 1. We utilized Matplotlib, a Python library, to visualize it. The red dots are the negative and the blue ones are the positive examples. As we can see in Figure 1 there are four

regions in which we can classify our data, two regions for each classification. Also, it is notable that our data are not linearly separable in the 2D dimension thus to have a good performance we need to choose a non-linear classifier such as SVM.

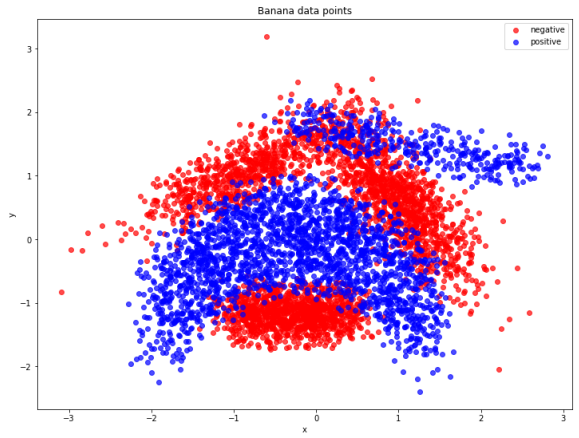

Fig. 1. Banana dataset with positive and negative labels.

For this experiment, we have used a Python library called Scikit-Learn. It has an SVM implementation with all those kernels mentioned in section II-B and we could prototype it without any problem. We let some default hyperparameter values from this library to compare each model with different kernels. Our regularization hyperparameter C is 1.0; it controls the cost of misclassified points. The hyperparameter γ is calculated with the Equation 5:

$$\gamma = \frac{1}{(n\_features)(Var[X])} \tag{5}$$

This hyperparameter controls the coefficient of the polynomial, sigmoid, and RBF kernels. For the polynomial kernel, its degree is set to three as the default value of the library. Also, as described in section II-A we need to vary the sigmoid kernel hyperparameter using 1, 0.05, and 0.01, we will represent each variation with a label on it.

Setting all of this, we can check our accuracy results for each kernel and each number of folds in Table I, also we can check each kernel out of sample error (Loss) in Table II. The best and worse results can be seen in each Table and with this, we can see that the best results are achieved from the RBF kernel and the worse are from sigmoid-0.01. For this specific dataset, it seems that the sigmoid kernel is not a good choice since it does not fit well on our data.

In Figure 2 and 3 we plot the support vectors and the decision surface with its margin of the RBF and sigmoid-0.01 models respectively. We can see that the RBF decision surface separates the data much better than the sigmoid surface.

To compare the RBF results, we try the KNN classifier; this classifier has a pretty simple strategy but yet it yields good results. This model idea is to get the k-nearest neighbors from a data point and classify it as the majority vote from their classes, with this it creates a decision region based on

TABLE I
ACCURACY TABLE

Models	K-Folds		
	2_folds	*5_folds*	*10_folds*
sigmoid-1.0	0.343396	0.341887	0.339623
sigmoid-0.5	0.329245	0.336415	0.335849
sigmoid-0.01	**0.289434**	0.290000	0.290377
rbf	0.902075	0.901509	**0.902264**
linear	0.551698	0.551698	0.551698
poly	0.632830	0.637736	0.637170

[a]Best and worst on bold.

TABLE II
LOSS TABLE

Models	K-Folds		
	2_folds	*5_folds*	*10_folds*
sigmoid-1.0	0.656604	0.658113	0.660377
sigmoid-0.5	0.670755	0.663585	0.664151
sigmoid-0.01	**0.710566**	0.710000	0.709623
rbf	0.097925	0.098491	**0.097736**
linear	0.448302	0.448302	0.448302
poly	0.367170	0.362264	0.362830

[a]Best and worst on bold.

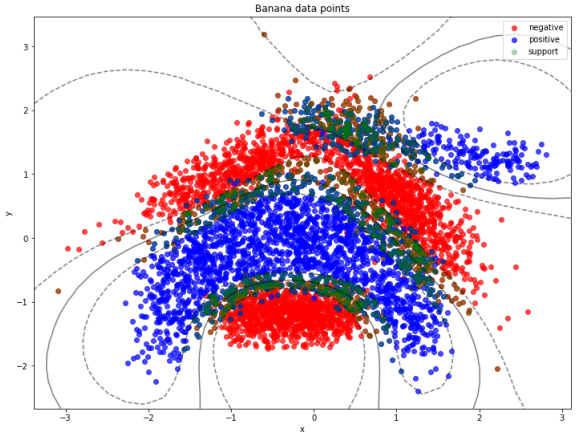

Fig. 2. Banana dataset with support vectors from best performance rbf kernel function.

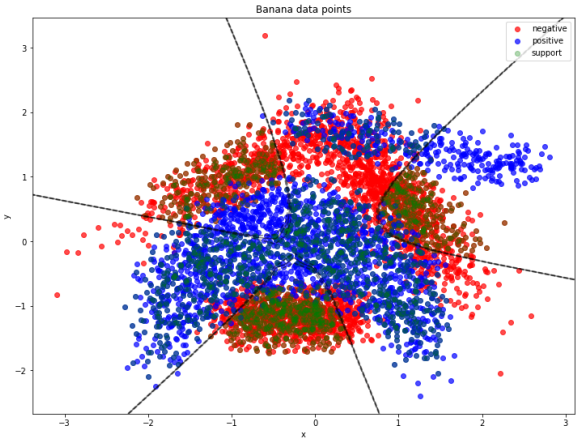

Fig. 3. Banana dataset with support vectors from worst performance sigmoid kernel function.

this classification. We have used an implementation of the KNN classifier from the same library as the SVM. We set our number of neighbors k to be 5 as in default, we changed its metric to euclidean distance and we gave a large weight for neighbors which are close to the data point. Also, we used cross-validation with 10 folds to ensure our results and on Table III we can check our accuracy and loss compared with SVM with RBF kernel.

TABLE III
KNN COMPARISON

Models	***Accuracy***	***Loss***
RBF	0.902264	0.097736
KNN	0.890377	0.109622

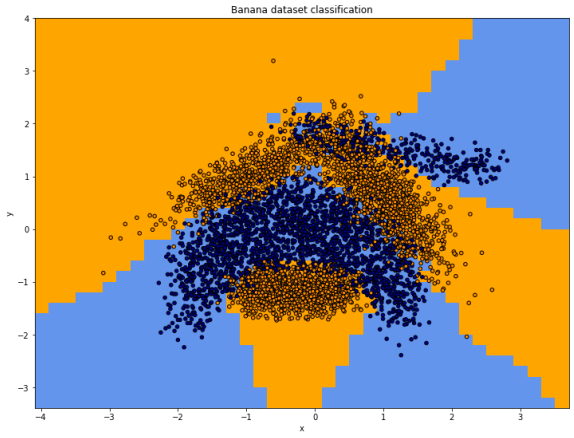

Fig. 4. Banana dataset with KNN classifier.

The SVM and KNN as a classifier have a good similar performance. This is achieved due to the good non-linearity data classification of both models.

III. Convolutional Neural Network Experiment

A. Experiment details

In this experiment, we are asked to classify handwritten digits using CNNs. For this we use the MNIST dataset; it consists of 70,000 images of handwritten digits from 0 to 9 in which 60,000 are split into train and 10,000 into test parts. Those images dimensions are 28 x 28 pixels and have only one color channel, the greyscale. With this data, we need to build a convolutional neural network able to have good accuracy on digit classification. Also, we need to plot a graph showing our model accuracy over epochs until we get maximum accuracy and we need to try different numbers of epochs from 1 to 5 and discuss its results. To test this architecture, we are going to fix some hyperparameters regarding the Pooling Layer and the Convolution Layer. As mentioned in the literature of CNN [5] we need to set the number of filters as a power of two, more specifically we set as 8, the kernel size we set as 3x3 matrix, and the pooling size for the pooling layer we set it to be 2 x 2.

B. Convolutional Neural Networks

Convolutional Neural Network is a class of deep neural networks most commonly applied to analyzing images. It is an architecture built with many layers such as Convolution Layers, Pooling Layers, and Fully Connected Layers. We can see an illustrated version of a simple CNN architecture in Figure 5.

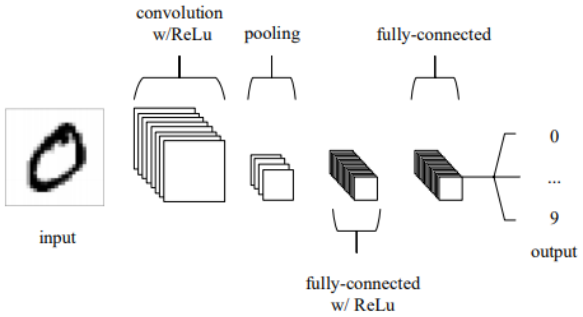

Fig. 5. Simple CNN architecture [5].

Convolutional layers are meant to extract features from our image inputs. These layers reduce image dimensionality while they keep image relations with it. They do this filtering the image and summarizing adjacent pixels into one value. Because pixels from images only have relationships with its neighbors, this is a good way of reducing complexity while maintaining information.

Pooling layers are used to reduce the spatial size of the image. It helps to deal with overfitting problems since it reduces the number of parameters our model will need to compute, decreasing in this way its complexity.

Fully connected layers are used in the end to classify our images. After all layers of reduction, our complexity is low compared with the start and we can construct a fully connected layer (often called dense layer) without paying all the price for this task.

C. Results

We can visualize a sample of MNIST for each digit class in Figure 6. With this dataset we trained over five epochs using an architecture described in section III-A we can check its summary in Figure 7. It is good to mention our choice of optimizer and loss for the dense layer; we followed a guideline on CNN literature to use Adam [6] as our optimizer and categorical cross entropy as our loss since we have multiple classes to predict.

After creating our CNN, we trained over five epochs to see its performance. In Figure 8 we can see its mean accuracy and cross entropy loss over epochs using cross-validation with 10-folds. Our first epoch has some different values for accuracy and loss, hence a high variance comparatively with other epochs. We can see in Figure 8 our model tendency to maximum accuracy and minimum loss, also we can see

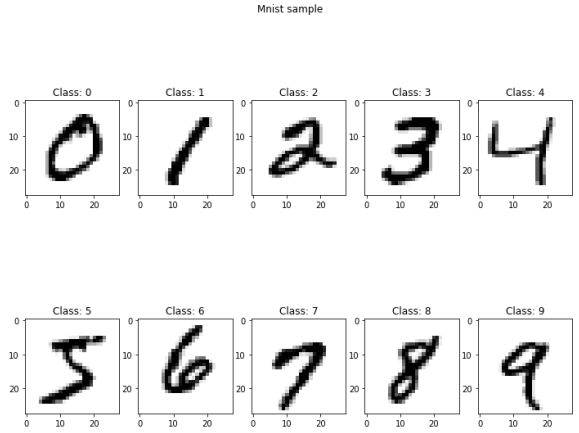

Fig. 6. Mnist sample for each class.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 8)	80
max_pooling2d (MaxPooling2D)	(None, 13, 13, 8)	0
flatten (Flatten)	(None, 1352)	0
dense (Dense)	(None, 10)	13530

Total params: 13,610
Trainable params: 13,610
Non-trainable params: 0

Fig. 7. CNN Layers summary parameters

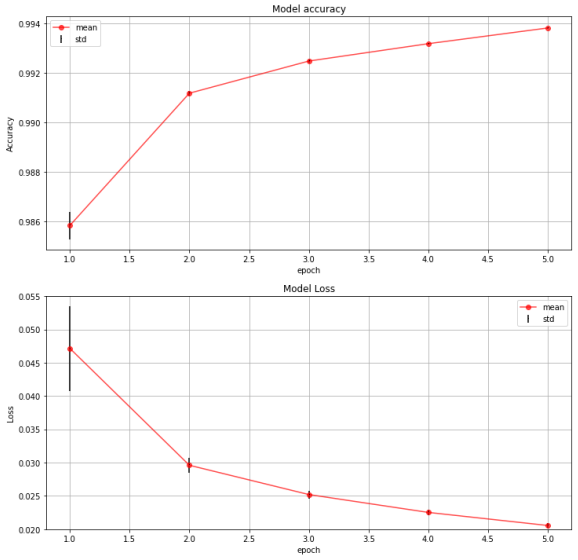

Fig. 8. Accuracy and loss of CNN model over epochs on training data.

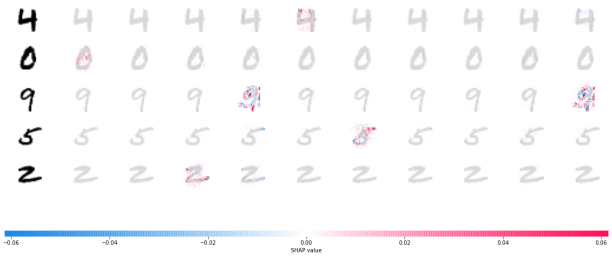

Fig. 9. Shap values in the CNN model for digit classification.

that our variance is decreasing when the number of epochs increases.

Since our loss is about our in sample loss, we can say that we are overfitting our model with this approach. This is explained because of the number of layers that we choose to be part of our architecture. We did not put many layers such as convolutional and pooling layers which could help our regularization problem. So with our architecture, we are trying to get too much information from the training set because we are using only a single layer of each type for this example.

Even though we had good accuracy and loss values in the validation set, more specifically 0.988 for accuracy and 0.036 for our loss. This is due to the good performance of this technique for image classification.

After all of that, we want to check how our model is predicting each digit correctly. For this, we will use a Python library called shap. It implements the SHapley Additive exPlanations (SHAP) technique that is a game-theoretic approach to explain the output of any machine learning model [7]. In Figure 9 we can check an explanation for our digit recognizer model. To read this image we need to explain how to do it. In this Figure the digits on the left are our test images; the digits with opacity are possible output predictions of the current Figure from 0 to 9 indicated with its position; red pixels increase the model's output while blue pixels decrease. For example, the digit 0 has multiple red pixels on its opaque picture on position 0, so it is correctly classified.

We see in Figure 9 that our CNN classifies most of those examples correctly, but we can see that it makes some mistakes when checking other classes as we look for the red dots in positions different from its classification.

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 8)	80
max_pooling2d_2 (MaxPooling2	(None, 13, 13, 8)	0
conv2d_3 (Conv2D)	(None, 11, 11, 64)	4672
max_pooling2d_3 (MaxPooling2	(None, 5, 5, 64)	0
flatten_2 (Flatten)	(None, 1600)	0
dense_2 (Dense)	(None, 10)	16010

Total params: 20,762
Trainable params: 20,762
Non-trainable params: 0

Fig. 10. Model summary with two convolutional and pooling layers.

So because of this we also tried to modify our architecture to see how it would improve our classification, for this we put one more convolutional and pooling layer before our fully

connected to try to identify more features within the image. This was recommended on CNN literature for classifying images as well as setting a power of two for the number of filters greater than the previous layer, so our new convolutional layer has 16 filters. We can see a summary of our new architecture in Figure 10.

We can see the improvement of classification in Figure 11, the red dots are more explicit on the correct classification position and the blue ones tell us why some classes were not chosen. Also on our validation set, we got 0.996 for our accuracy and 0.011 for our loss which is better than our previous model architecture.

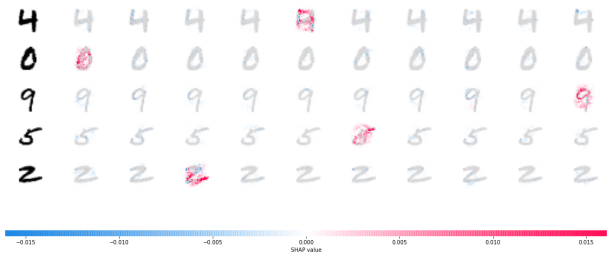

Fig. 11. Shap values using new CNN architecture.

IV. Conclusions

To conclude we can say after those experiments that SVMs are a good and powerful choice to deal with a classification problem with non-linearly separable data. We also see that we have other options for this kind of classification as well such as the K-Nearest Neighbors classifier. Those two yields good results on the kind of data that we used.

In the second experiment, we showed the details of constructing an architecture to handle this task and explained how you should do it. In the end, we used a method to explain how the CNN model is predicting its output using the Shap method. With this last one, we were able to see some indecision about our model on choosing its output so we upgraded our old architecture into a new one. After this upgrade, we saw better performance on our digit classification task.

References

[1] KEEL, Knowledge Extraction based on Evolutionary Learning, Banana Dataset. https://sci2s.ugr.es/keel/dataset.php?cod=182#sub2, 2020.
[2] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010.
[3] Cortes, C. & Vapnik, V., 1995. Support-vector networks. Machine learning, 20(3), pp.273–297.
[4] Cunningham, Padraig & Delany, Sarah. (2007). k-Nearest neighbour classifiers. Mult Classif Syst.
[5] O’Shea, Keiron, and Ryan Nash. ”An introduction to convolutional neural networks.” arXiv preprint arXiv:1511.08458 (2015).
[6] Kingma, Diederik P., and J. Adam Ba. ”A method for stochastic optimization. arXiv 2014.” arXiv preprint arXiv:1412.6980 434 (2019).
[7] Lundberg, Scott M., and Su-In Lee. ”A unified approach to interpreting model predictions.” Advances in neural information processing systems. 2017.