Cover page for answers.pdf

CSE512 Spring 2021 - Machine Learning - Homework 4

Your Name: Venkata Ravi Teja Takkella

Solar ID: 113219890

NetID email address: venkataravite.takkella@stonybrook.edu

Names of people whom you discussed the homework with: NA

(1)

(1.1) Given to train a Linear SVM model.

It consits of n points & given m support vectors are obtained after training on the entire dataset. Now need to show the LOOCV error is bounded above by m/n.

First the error for an LOOCV is taken as

$$= \frac{\text{no. of wrong classifications among all folds}}{\text{Total no. of folds.}}$$

Here no. of folds = n ⇒ total no of data points as it is LOOCV.

So for LOOCV in SVM which is Linearly Seperable. We can take 2 cases

(1) Picking non-support vector as the test set?

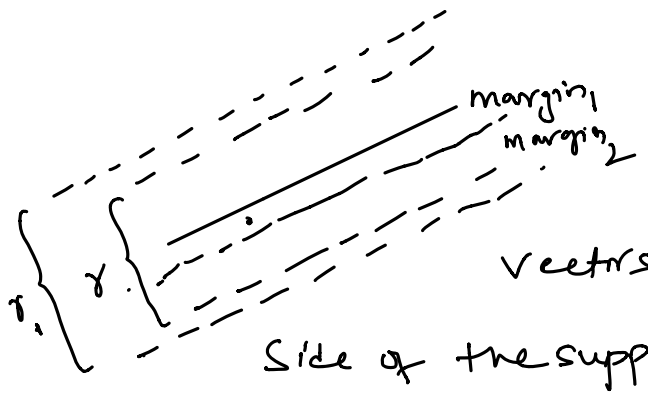
In this case, the margin width & the seperation line doesn't change as they are totally dependent on the support vectors & they are still intact in the training data.

So the test data which is a non-support

vector will still be classified correctly in the way it was classified in Linear SVM.

(2) Picking a support vector as the test set:

While picking a support vector, this might change the seperating line & width.



Here since it's given as linearly seperable. All the support vectors initially placed on the correct side of the supporting line but on the wrong side of the margin. If the support vector taken out as test dataset is so close to the seperating line. Removing that would move the line bit more towards it's class.

And classifying this test data would give out a wrong class. And if all the SVs are in the same situation. We can end up with m failed classifications.

Hence with these both scenarios, we'll get the max. error when all SV's come up on the wrong side.

then error = $\frac{m}{n}$ → No. of wrong classification
→ Total no. of datapoints.

So if any of those SVs still come up on the right side.

then error $< \frac{m}{n}$.

So LOOCV error is always bounded above by m/n.

(1.2) Here we are asked to take a general kernel instead of Linear kernel.

And also given that the data is linearly seperable in the high dimensional feature space corresponding to the kernel.

So here the main change which we do is that instead of the seperating line of the data points → we take a hyperplane in those dimensions. And rest of the concepts still work the same. The concept of hyperplanes & the effect of LOOCV when a support or non-support vector is taken as the test set works the same.

Because the concepts of line are just replicated to the hyperplane with higher dimensions.

So as the no. of support vectors are given as m in total of n data points.

the LOOCV error again gives it's error as

$$= \frac{\text{no. of wrong classifications among all folds}}{\text{Total no. of folds.}}$$

The denominator is n again. And the numerator brings out n at maximum, if all the support vectors are on the wrong side of the hyperplane when they are removed & used as test data set.

And similarly even after removing one & use them as test data, still they end up on correct side of the hyperplane. the error for that classification would be 0.

So basically the error of LOOCV is still bounded above by m/n.

2)

Setup)

As mentioned in the assignment, I've submitted my Google colab notebook mentioned which is under the name **Homework4Q2.ipynb.**

And after extracting the **HW4_q2** zip, please place the **Q2.py** python file in the same folder for the python script to run perfectly.

And all the pip installs are mentioned in a bash script mentioned in the submitted folder as **configsQ2.sh**

2.1)

Here we were asked to find the accuracy and confusion matrix on both train and test data sets. After implementing the XGBoost, the metrics are shown below.

Accuracy score for Train dataset : 0.9045483861060778

Confusion matrix for Train dataset :
[[23672 2060]
[1048 5781]]

Confusion matrix : Normalised for Train dataset :
[[0.91994404 0.08005596]
[0.15346317 0.84653683]]

Accuracy score for Test dataset : 0.8696640255512561

Confusion matrix for Test dataset :
[[11663 1350]
[772 2496]]

Confusion matrix : Normalised for Test dataset :
[[0.89625759 0.10374241]
[0.23623011 0.76376989]]

2.2)

Here we were asked to perform K-fold using k= 10 for XGBoost and take the the parameters of the fold which got the highest accuracy and use those parameters to train on the entire test data set.

The parameters I used for tuning the classifier for each fold are:
'Learning_rate'
'Max_depth'
'Min_child_weight'
'Gamma'
'colsample_bytree'

When performing the K-fold, the highest accuracy I obtained was
0.8774570024570024

And the parameters used for the same are:
{'learning_rate': 0.15, 'max_depth': 12, 'min_child_weight': 1, 'gamma': 0.2, 'colsample_bytree': 0.7, 'objective': 'binary:logistic', 'use_label_encoder': False}

And when I used these parameters on the entire test dataset, the Accuracy and confusion matrix obtained are :

Accuracy score for Test dataset :
0.8756218905472637

Confusion matrix for Test dataset :
[[11761 1351]
[674 2495]]

Confusion matrix : Normalised for Test dataset :
[[0.89696461 0.10303539]
[0.21268539 0.78731461]]

3)

Setup)

As mentioned in the assignment, I've submitted my Google colab notebook mentioned which is under the name **Homework4Q3.ipynb.**

And after extracting the **HW4_q3** zip, please place the **Q3.py** python file in the same folder for the python script to run perfectly along with hw4_utils.py and detect.py..

And all the pip installs are mentioned in a bash script mentioned in the submitted folder as **configsQ3.sh (Please check on the initial cd command mentioned in the bash script, which is based on my colab)**

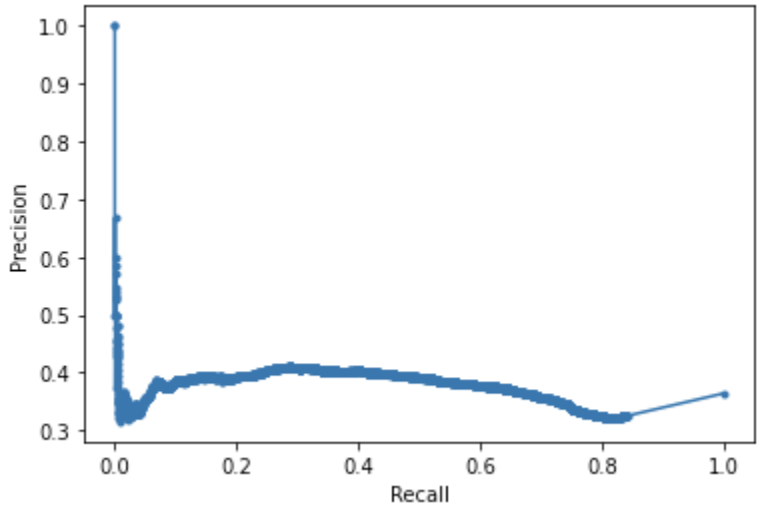
3.1)

Here we were asked to train a Linear SVM classifier using the train data and provide the AP using compute_mAP() and plot the precision recall curve by testing on the validation data.

I've trained on the entire training data set available and the testing is also done on the entire validation data available.

The mAP obtained after testing on the validation data is
5.6008932006079704e-05

And the precision recall curve looks like as below :



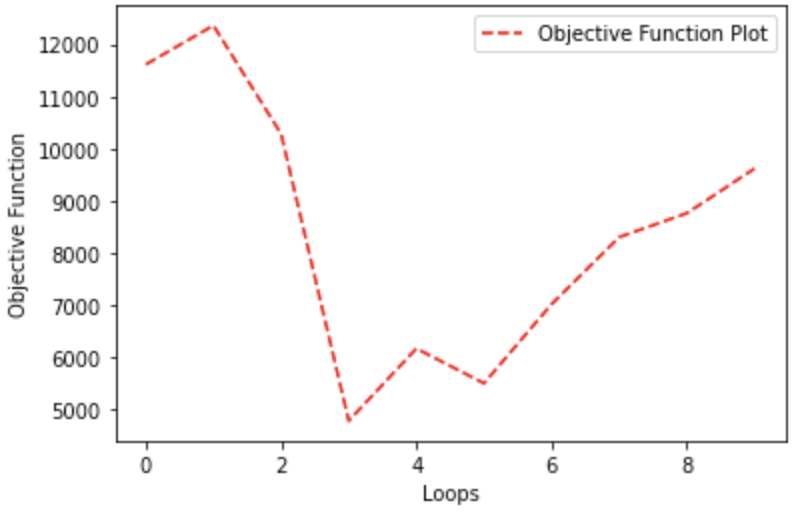
3.3)

I've used the hard negative mining algorithm built in 3.2 and ran it for the mentioned number of loops and parameters. I've made sure that the number of negative samples doesn't cross 10,000 at the maximum. I've trained the classifier by adding hard negative samples for 10 loops. The method behind the algorithm is, I've used the detect function to pick random images from the validation data as mentioned and brought out the predicted rectangles. Then using the get_iou function, I've found out the overlap and taken all the rectangles which are below the threshold (0.3). So these are the hard negative samples which are added for each iteration and the classifier is trained on the same data.

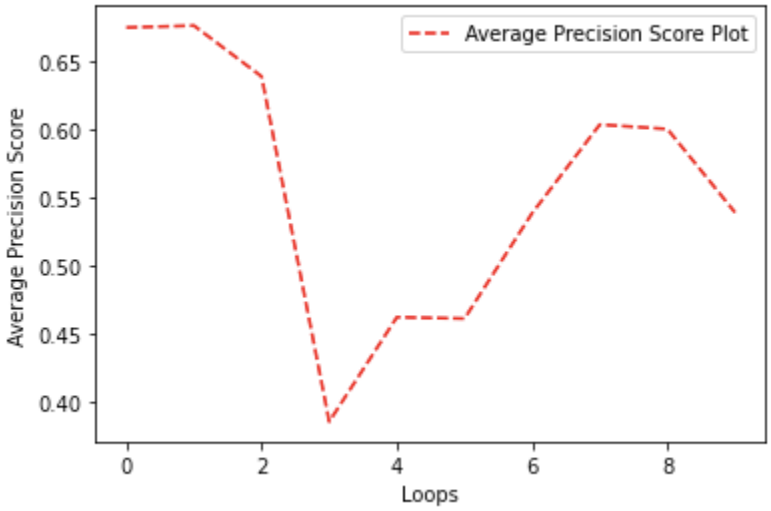
The objective function I've used =

$$obj^{(t)} \leq obj^{(*)} \leq \underbrace{obj^{(t)} + C \sum_j \bar{\xi}^{i_j}}_{u^{(t)}}$$

So after running on the above parameters, the plots for the Objective Function and average precision score looks like with X-axis being the loop number:



Loops



The dip in the curves (around 3rd loop) can be explained as this might've taken the classifier to reach the optimum region and from there, adding on hard negative samples just increases both the objective function and AP.

3.4)

As mentioned, I've submitted my npy file generated using the generate_result_file and the provided AP value is:

0.0001032

P.S : Please don’t hesitate to contact me in case of any setup issues of my code.