CSDS440-21Summer-P1 Final Report on

Online Learning on Large-Scale Dataset by Kernel Tricks and other Approximation Methods

Ellis Wright, Jacob Prusky, Michael O'Brien, Sohan Bhawtankar, Zhou Yang

Overview

Introduction

In this report, we investigated the topics of online kernel learning in different perspectives and its practical applications for different real-world tasks including image recognition, stock prediction and sports analysis, etc.

While batch learning could have access to all training data, due to the nature of the learning task and the dataset, it might be impossible to collect desired data at the very beginning of the training process. For example, the stock information keeps updating every millisecond and much data for advertising tasks and recommendation tasks are generated by continuous interactions with users. This created the initial need for online learning which could update the leaner model with the newly incoming training samples without to re-train the entire model.

Significance

However, due to the application scenarios of online learning, it is easy to encounter an excessively large-scale dataset to process, which results in expensive, even infeasible, cost of computational resources.

Most online learners use a set of support vectors(SV's) in memory for representing the kernel-based predictive model[2]. One major challenge of this kind of approach is that, when a new incoming training instance is misclassified, the SV set will be expanded. Therefore, for large-scale dataset, the expansion of the set of SV is unbounded which makes the proper control of computational resources becomes infeasible.

Many approaches have been introduced to address this issue for large-scale applications, one major direction is to limit the size of SV set[1] by different budget control strategies including:

1. SV Removal[1,3,4]
2. SV Projection[4,5]
3. SV Merging[6]

The shortcomings of those budget strategies mainly come from these issues:

1. For some existing algorithms proved to be efficient, the accuracy is not always satisfactory.
2. The computational resource consumption is still high.
3. It brings more communication overhead costs in distributed systems with large-scale dataset.[7]

Besides the above approaches, there is another direction to address the issue which is also our main investigating target: using kernel functional approximation techniques to reduce the computational resource consumption:

1. To adopts the random Fourier features and Nystrom method for approximating kernel.[2]
2. To apply reparameterized tricks on random Fourier features for approximation.[8]
3.

One of the main advantages of the functional approximation is that it achieves the efficiency of kernel learning tricks while it could also utilize existing efficient linear online learning algorithms.

Individual Report: Zhou Yang

Introduction

For the nature of the service, intuitively, recommendation systems and the anti-spam filters are always in the need for high performance online learning algorithms for large-scale datasets. Thus my survey goes into the different kernel approximation methods and some applications of large-scale online learning in the security field like to detect the malicious URL.

Approaches Investigated

Online Kernel Learning with Reparameterized Random Feature[8]

Reparameterized Random Feature online kernel method is one of the main algorithms examined in this survey. It applies a similar philosophy of Random Fourier Features to utilize the kernel tricks to make approximations of the computations.

RRF further extends the RFF to use random features in kernel approximation which reduces the computations compared to the method of considering all features. In addition, the random features also grants the researchers more freedom to control the level of appromization by adjusting the numbers and distributions of the features to be selected.

After the random features, RRF transforms the kernel parameter learning task into an optimization problem across the class of parameterized Reproducible Kernel Hilbert Spaces(RKHS). By doing so, the kernel parameters are also accurately approximated.

The implementation of RRF is more complex compared to Random Budgeted Perceptron(RBP) and the original RRF. It supports binary/multi-class classification and regression. In addition, it uses Stochastic Gradient Descent(SGD) for updating the weights.

One thing to note is that the RRF is sensitive to the choices of hyperparameters and the ideal values of hyperparameters can only be determined in a range instead of exact values. It might need to run multiple rounds to determine the best-performance values.

Random Budgeted Perceptron[9]

The random budgeted perceptron method is a classic method to address the infeasible memory usage and intensive computation issues for large-scale online kernel learning using the budget method.

Researchers first built a Perceptron classifier called Shifting Perceptron Algorithm(SPA). The key idea of shifting Perceptron is shifting the decision boundary hyperplanes. For the misclassified example pairs (x,y), just like the original Perceptron update rule, the signed instance vector yx will be added to the old weight vector. However, the difference is that the algorithm will scale down the old weight so that the importance of the early update stages will be diminished.

The SPA creates a bound of the growth rate of the norm of the algorithm's weight vector as:

$$|w_{k+1}| = e * sqrt(\lambda + k + 2/2\lambda + 1)$$

It is obvious that such bound is determined by the values of λ.

Thus, although SPA provides a bound, for some special values of the λ, the support vectors added to the memory will increase exponentially with the number of newly misclassified examples. Thus, a natural way to 'hardly' bound the expansion of the weight vector set is to set an upper limit of the total

number of support vectors. The Random Budgeted Perceptron Algorithm (RBP) is introduced for such purposes.

On the basic logic of SPA, when a new support vector is about to be added in to the weight vector set, RBP will check the current size of weight vector set: if the size is not exceeding the budget, the new support vector will be added normally as SPA, otherwise, if the budget is reached, while the new vector is added, one randomly selected old weight vector will be removed from weight vector set. Thus the memory usage budget will be maintained.

Both SPA and RBP are modified from the Perceptron model with relatively simple modifications. They are easy to implement and the underlying logic is concise. The mathematical proof of bounding and the expectation of the mistakes occurring are also complete.

Application:Malicious URL detection[10]

Originally my survey aimed to investigate the applications of online kernel learning in fields of recommendation systems and anti-spam. During the investigation, the task to detect of the malicious URL shows an interesting attribute that comparing to continuous-valued or categorical features in recommendation system and word embedding features for spam emails/messages, the contents of URL could not be processed directly by NLP tools while they are also structured formatted data which contains some information that is meaningful for the functions of URLs. For this reason, the actual feature space of URL detection tasks will be relatively large even though the size of examples might not.

To extract the structured information from raw URLs, the researchers first set up some rules based on human knowledge like using website rankings like Alexa rank to evaluate and classify the domains of the URLs complying with other lexical methods to extract the processed features.

Then for processed URLs, there are 2 issues to address: 1. How to select the independent features that could efficiently determine if the URL is benign or malicious. 2. The results dataset is still too large. Researchers apply the novel Distance Metric Learning(DML) approach to address issue 1.

The objective of DML is to minimize the distances between the data points in the same class and maximize the distances between the data points in the different class.

By solving the Linear Programming Problem formulated by DML, the algorithm could produce 2 matrices to compute the DML approximized features efficiently.

For problem 2, the paper applies Nyström method which is a kernel method also used in [2] to make approximations on kernel matrices for less computational consumption.

The key contribution of the paper focuses on the feature engineering techniques described above. The proposed system converts the original high-dimensional features into lower dimensions using kernel trick and DML methods reducing the resources needed for the task. For the later classification model, the paper just applied popular methods like k-NN, L-SVM, CNN etc.

One thing to notice is that the k-NN method with DML feature tricks achieves the best performance based on the table 5 in the paper. One of my guesses is that since both k-NN and DML are learning from the distance relations in the dataset, the information extracted by the DML method could have larger importance in the k-NN model.

Implementation

2 algorithms are implemented in this report:

1. Online Kernel Learning with Reparameterized Random Feature (**RRF**).[8]
2. Random Budgeted Perceptron.(**RBP**).[9]

The Random Budgeted Perceptron algorithm is implemented as a baseline budget model to compare with RRF since it is one of the basic algorithms using the SV removal approach.

In original paper for RRF, the RRF using hinge loss function achieves best performance, thus I use same loss function and $\boldsymbol{\lambda}$ is in the range of $[2^{-4}/M,...,2^{16}/M]$,$\boldsymbol{\eta}$ is in $[10^{-5},3 * 10^{-5},10^{-4},...,]$ and $\boldsymbol{\gamma}$ is in $[2^{-8},2^{-4},2^{-2},2^{0},2^{2},2^{4},2^{8}]$ where M is the size of the training dataset.

All algorithms are implemented with numpy for math operations and sklearn for utility tools. Some of the evaluation metrics functions are modified from programming assignments.

During the evaluation of RRF algorithm, I noticed that the hyperparameters:$\boldsymbol{\lambda}$,$\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$ could not be easily determined. For best performance, they are determined by training and testing in sub-optimal ranges derived by [8] and then picked the best values. However, this approach requires repeated computations and violates the initial purpose to save the computational resources. Thus, I implemented an extension to RRF such that:

1. To take 10% of the training set as the pre-training dataset.
2. To split the pre-training dataset by 7.5:2.5 for training:testing.
3. Evaluate the model on the smaller pre-training dataset.
4. Uses the best hyperparameters as the hyperparameters for training on the whole dataset.

Note, to measure the performance of the budget approach and approximation approach, I used the pre-implemented Perceptron model in sklearn as the **baseline**. Although this model will show in the evaluation section, it is not implemented by me.

Datasets

3 Datasets are used for evaluating the algorithms I implemented.

1. Netflix Prize Dataset
2. SMS Spam Dataset
3. Loan Dataset

To unify the format of 3 datasets, following preprocessings are applied on raw data:

1. SMS Spam Dataset contains the raw SMS text contents as the feature. The stop words and non-character words are removed from the raw data. Then the text contents are tokenized and vectorized using nltk tool kit. A generated 2500-length feature list is used for learning.
2. Since Randomized Budget Perceptron is an algorithm that originally for binary classification tasks and other 2 datasets are binary datasets, the labels (rating numbers) of Netflix Prize Dataset are mapped to binary as “Recommended-1/Not Recommended-0” and also simulates the scenario that if the system should recommend specific movie to specific user.
3. The feature values in Loan Dataset are normalized before the training.

Evaluation and Discussion

To evaluate the classifiers implemented, the following metrics are recorded:

Mistake Rate: Incorrectly labeled predictions / all samples.

Training Time: The average time to run a complete training process.

The comparison evaluation:

Table 1-1 RRF

Dataset	Mistake_Rate	Training Time (Seconds)
Loan	14.1	0.7
Spam	12.0	25.3
Netflix	24.1	2.6

Table 1-2 RBP

Dataset	Mistake_Rate	Training Time (Seconds)
Loan	16.0	0.03
Spam	13.7	0.04
Netflix	24.1	0.1

Table 1-3 Baseline

Dataset	Mistake_Rate	Training Time (Seconds)
Loan	14.9	NA
Spam	11.8	NA
Netflix	24.1	NA

Note: Since Baseline is not implemented by myself and might be optimized, the training time is not recorded for comparison.

It is expected that, since RRF and RBP are using approximation methods to reduce computations and memory usage, they have to trade off some accuracy compared to the original Perceptron baseline which holds all information and does not use approximation methods.

We can see that even though RRF has lower mistake rates than RBP, the training time of RRF is worse than RBP. Due to the design of RRF, the number D of random features used for approximation is critical for its training time, since a larger D leads to larger matrices and the larger matrices makes the matrix operations slower. By comparing the training time of 3 datasets we can see, although the spam dataset has only ⅛ of examples of Netflix, the Netflix dataset still has much less training time. This is because the Spam dataset has a 2500-length feature list and uses D=100 for random feature learning while the Netflix dataset has only 3 features. The trade-off between the larger D for better accuracy and longer training time could be observed by internal comparisons of 3 datasets and the cross comparisons to RBP.

Another guess is that, since the datasets for testing are not large enough due to the limitation of hardware, RRF methods spend some time on weight construction,memory operations and the matrix operations are not distributed by the larger number of examples. Thus the overhead cost causes the RRF to perform slower than RBP for the tested dataset. The differences between the training times might shrink as the scales of datasets grow.

All 3 algorithms do not have good performance on dataset Netflix, that might be due to the bad representations of the dataset or the linear models might not be suitable for this task.

The extension evaluation:
On 3 different datasets, I used the full dataset and the extension implementation(partial dataset) to select best values of hyperparameters and got the results as follows:

Table 2-1 Using Full Dataset

Dataset	$\boldsymbol{\lambda}$	η	γ	**Mistake_Rate**
Loan	2^16/M	10^-5	2^-2	16.0
Spam	2^16/M	10^-5	2^-4	13.7
Netflix	2^-4/M	10^-5	2^8	24.1

Table 2-2 Using Partial Dataset

Dataset	$\boldsymbol{\lambda}$	η	γ	**Mistake_Rate**
Loan	2^16/M	10^-5	2^-8	16.0
Spam	2^-4/M	10^-5	2^-8	13.7
Netflix	2^-4/M	10^-5	2^-8	24.1

From the Table2 above we can see that only using part of the dataset we could achieve a similar mistake rate with different values of hyperparameters. Although for Netflix dataset, no matter for full dataset or partial dataset, the mistake rate is not low enough, it might be due to the lack of dimensions of features of the dataset: the Netflix dataset only has 3 features thus the random fourier features and random reparameterized features could not benefit from the small D value.

But for the Loan and Spam dataset, we can conclude that we can use a smaller portion of the dataset to select the hyperparameters and apply it to the model of the whole dataset.

Individual Report:
Sohan Bhawtankar

Introduction and Algorithm

The algorithm I implemented classifies examples into binary labels based on a set amount of kernels (each kernel must output a value less than 1), and is based on the application of two online learning algorithms: The Perceptron algorithm and the Hedge algorithm. The perceptron algorithm learns a classifier for each kernel, and the Hedge algorithm assigns a weight to each kernel, to linearly combine their outputs and get a value for the label. This is a rather straightforward solution to classifying data with multiple kernels. This variant of the algorithm also uses a deterministic approach in updating the weights of each classifier. [1]

While multiple kernel learning normally is computationally expensive, this algorithm reduces the time and expenses by only having to scan through the entire training dataset once.

Implementation

For this implementation, I make use of four kernels, and use a constant value of 10 for every λ:

1. Gaussian Kernel: $e^{-||x-y||^2*\lambda}$
2. Exponential Kernel: $e^{-||x-y|| *\lambda}$
3. Inverse Multiquadric Kernel: $\frac{1}{\sqrt{||x-y||^2}}$
4. Cauchy Kernel: $\frac{1}{1+||x-y||^2*\lambda}$

The algorithm automatically bounds the number of mistakes, where

Number of Mistakes = $\sum_{t=t}^{T} I(q_t \cdot z_t \geq 0.5) \leq \frac{2ln(1/\beta)}{1-\beta} min_{1\leq i\leq m} g(k,l) + \frac{2ln(m)}{1-\beta}$

Terms:

1. $f_t = (f_{1,} f_{2,} f_3, f_4)$, where each f is a kernel classifier
2. $w_t = (w_1, w_2, w_3, w_4)$, where w corresponds to the weight of a kernel classifier
3. $q_t = (q_1, q_2, q_3, q_4)$, a normalized weight vector where each q corresponds to the normalized weight of a kernel classifier
4. $z_t = (z_1, z_2, z_3, z_4)$, an indicator vector where each z = I(y*f(x) < 0), where y is the true label of the example passed through the algorithm, and x is the example. I(C) returns either a 1 or a 0, a 1 representing that the kernel classifier did not correctly predict the label and that the classifier and weights must be updated.
5. βis a term used to discount a weight of a kernel, when it classifies an example incorrectly.

Training steps:

*This process repeats for every training example t

1. Predict the label of the example according to each kernel classifier in f_t
2. Receive the true label of the example, y_t
3. Find z_t

4. Update the weight of each classifier to w_t*β^{z_t}: note that the weight will only update if the classifier predicted wrong, because z_t would be 1. For this implementation, I set βto be 0.8
5. Update each classifier by z_t* y_t* k(t, ·): again, note that the classifier will only update if the classifier predicted wrong

Predicting steps:
*This process repeats for every example we are predicting, t
1. Predict the label of the according to each kernel classifier
2. The predicted label of the example will be sign(q_t·sign(f_t))

Libraries:
1. Pandas - To read csv files of data, and sort and separate the features and labels
2. Numpy - To transform examples into arrays
3. Math - In order to implement the kernel functions

Datasets
To evaluate the accuracy of classifying data for the algorithm I implemented, I used the following two datasets:

1. Banknote authentication dataset
    - This dataset is from the Machine Learning Repository of UC Irvine
    - The dataset contains data extracted from images of genuine and forged banknotes, and categorizes them as 1 if forged, and 0 if not
    - The algorithm will predict whether or not a banknote is forged depending on the data from an image of the note
2. Pima Indians Diabetes dataset
    - This dataset is from the National Institute of Diabetes and Digestive and Kidney Diseases
    - It measures different medical predictor variables for Pima Indian women and a label: 1 if they have diabetes, and 0 if not.
    - The algorithm will predict whether or not a patient has diabetes depending on these variables

Evaluation
Note: I will be using the same λ for each kernel. And for training, I will use a 1:1 ratio of positive and negative labels, and predict on 1000 examples. I use a constant gamma value, because with previous testing, I deduced that changing λ results in absolutely no effect on the prediction on these specific datasets.

Banknote

# Training Examples	λ	Accuracy
100	**10**	**0.994**
200	**10**	**0.996**

400	**10**	**0.996**
1000	**10**	**1.0**

Diabetes

# Training Examples	λ	Accuracy
100	**10**	**0.734375**
200	**10**	**0.8046875**
400	**10**	**0.87109375**
1000	**10**	**1.0**

As shown in the tables, as the algorithm learns from more examples, it does a better job of classifying the other examples. However, as seen in the Diabetes dataset, it is harder to classify data correctly just based on a fewer number of training examples. I hypothesize that this occurs because the dataset has more features than the banknote one, and requires more support vector machines in order to correctly classify the data. If I had more time, I could perhaps implement more kernel classifiers in order to produce more accurate results (maybe ten kernel classifiers instead of four). Also, instead of updating deterministically, I could try updating the kernel classifiers stochastically.

Extension and Evaluation

For my research extension, I took inspiration from the naive Bayes algorithm we implemented earlier, and chose to discretize the continuous values of the datasets and see whether or not the algorithm would improve. I did this by splitting the continuous values into 10 different bins. For the purpose of this implementation, I define a feature as continuous if it has more than two distinct values.

Banknote

# Training Examples	Bins	Accuracy
100	**10**	**.55**
200	**10**	**0.55**
400	**10**	**0.329**
1000	**10**	**0.55**

Diabetes

# Training Examples	Bins	Accuracy

100	**10**	**.7044**
200	**10**	**0.7513**
400	**10**	**0.84765625**
1000	**10**	**0.9986979166666666**

As seen in the tables above, discretizing the continuous values performs just a bit worse in the Diabetes dataset, but in the Banknote dataset, the algorithm performs significantly worse no matter how many training examples it takes. I theorize that an explanation for this discrepancy is that it is most likely due to the importance of the continuous variables in the dataset. In the diabetes dataset, the features include factors such as blood pressure, age, glucose levels, etc. High values for these features are usually directly correlated with diabetes, which is why putting them into ten different bins will allow for a similar accuracy. In the case of the Banknote dataset, I can observe the values of each feature and see that there is not much of a correlation between certain values of features and the labels. Discretizing the continuous features into bins would work against the accuracy of the algorithm in this case.

What this does show is that discretizing the features can be a valid step to take while training and classifying, but should not be done for every continuous feature. Instead, the algorithm should be able to select features to discretize based on the correlation of their values with the labels.

Conclusion

This project evaluates a rather straightforward approach to classifying data with multiple kernels. Implementing a combination of the perceptron and hedge algorithms allows for a less computationally intensive algorithm while training and classifying data. The algorithm only has to run through the training data once, before updating the weights and getting a final classifier. While simplistic, this algorithm does a great job of classifying data, reaching accuracies of over 90% quite easily.

References

1. Rong Jin, Steven C.H. Hoi, and Tianbao Yang. “Online Multiple Kernel Learning: Algorithms and Mistake Bounds”.
2. Yanning Shen, Tianyi Chen, and Georgios B. Giannikis. “Random Feature-based Online Multi-kernel Learning in Environment with Unknown Dynamics”. Journal of Machine Learning Research 20 (2019) 1-36. 2/19

Individual Report: Ellis Wright

Overview:

The algorithm I explored is an Extreme Learning Machine. The datasets I have used for binary classification are the well-known iris data (without Iris Virginica), the loan dataset, the spam data set, and the volcanoes data set from class. An Extreme Learning Machine is a special type of Single-Layer-Feedforward-Network (SFLN). One of the main issues with neural networks is the time it takes to learn the weights. Extreme Learning Machines speed this process up by removing the time bottleneck during backpropagation. There are two main parts to an extreme learning machine, the input weights and the output weights. Extreme learning machines take one hyperparameter, L. L is the number of hidden nodes in the hidden layer. The input layer is fully connected to the hidden layer. Because there is no backpropagation, training the network is quite different- we only need to solve for the output weights. We solve for the output weights, $\boldsymbol{\beta}$,by setting up a linear system and solving for $\boldsymbol{\beta}$ using the Moore-Penrose generalization of the inverse. This is the same as minimizing the squared loss because we are solving the for the least squares line (LS). Using the rest of the data points, or using the data in an online setting, we can apply a recursive least squares algorithm (RLS) to update $\boldsymbol{\beta}$ as we experience more data.

Algorithm

Let D be a full data set. $D \in \mathbb{R}^{mxn}$
Let Z be the first L data points of the training set. $Z \in \mathbb{R}^{Lxn}$
Let $\boldsymbol{\beta}$ be the output weights. $\boldsymbol{\beta} \in \mathbb{R}^{Lx1}$
Let W be the input weight matrix. $W \in \mathbb{R}^{Lxn}$
Let σ be the biases. $\sigma \in \mathbb{R}^{L}$
Let Y be the measured labels. $Y \in \mathbb{R}^{Lx1}$
Let g be the activation function, (any infinitely differentiable function, for our purpose- assume sigmoid).

The algorithm can be summarized as followed:

1) initialize W and σ with random values
2) calculate the hidden layer output matrix , $H = g(WZ^T - \sigma)$
3) solve the linear system $H\boldsymbol{\beta} = Y$ by calculating the Moore-Penrose inverse of H.
    a) $\boldsymbol{\beta} = (H^TH)^{-1}H^TY$
4) initilize $M_0 = (H^TH)^{-1}$ and $\boldsymbol{\beta}_0 = \boldsymbol{\beta}$
    a) for the rest of the data points $\{\mathbf{x}_i, y_i\}$ for i = L+1,...m apply the RLS algorithm[1]
        i) $M_{i+1} = M_i - (M_i h_{i+1} h^T_{i+1} M_i)/(1 + h^T_{i+1} M_i h_{i+1})$
        ii) $\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i + M_{i+1} h_{i+1}(y_{i+1} - h^T_{i+1}\boldsymbol{\beta}_i)$

There is not a consensus on how many hidden nodes to use, yet this will bound how many features the classifier can pick up on. Note: it is not required that Z has to contain L instances but the theory shows that the network can learn L features this way.[2]

Results

The metrics used to evaluate the performance are the accuracy and time to run. I did not make other algorithms so I did not compare these metrics so they can be used as benchmarks for further research.

Here are the learning curves for the Iris and Loan data sets. The loan data set took 4.682379961013794 seconds to run and the iris data set took 0.0425570011138916 seconds to run. The accuracy is shown in figures 1 and 2, with the loan dataset gettin 83% and the iris data set hitting 100% classification accuracy.

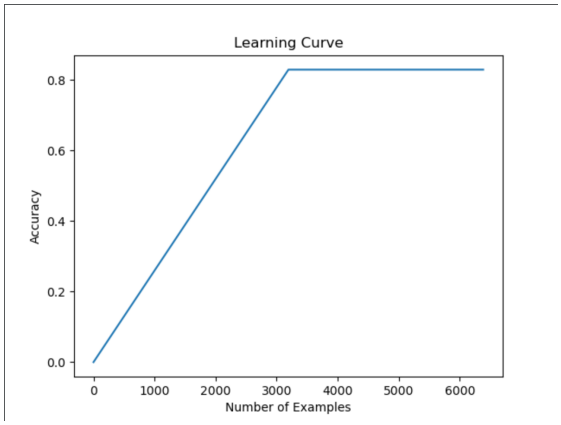


Figure 1: Loan dataset learning curve

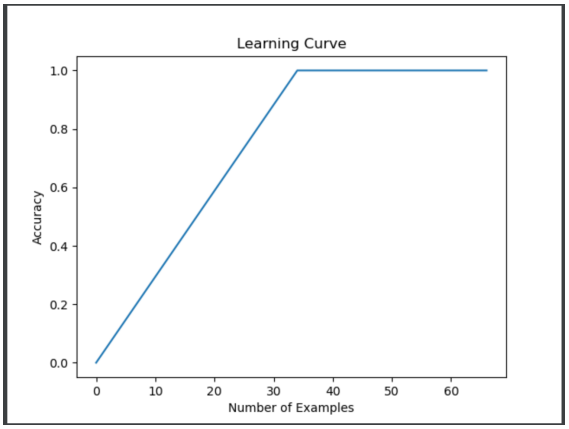


Figure 2: Iris dataset learning curve

Figures generated using matplotlib.

Discussion

Extreme learning assumes the data is in continuous format. This will help guarantee that an inverse will exist for the linear system. Unfortunately, I found this out way too late so on some datasets the algorithm cannot predict better than always guessing positive, if most data is labeled positive. I decided to center my research extension around this by adding tiny amounts of gaussian noise to nominal data so I could extend the algorithm to more datasets. When running on the loan dataset, the algorithm still cannot calculate the inverse of the matrix, so my implementation fails. This also happened on the volcanoes dataset from class. This is the part I was most confused about because I don't see how we can guarantee that there will always be a solution. One hypothesis I have is that the spam data set is not linearly separable. Another, possibly more plausible hypothesis is that I incorrectly implemented the algorithm. Huang claims that because the data has been generated by a continuous probability distribution, it is safe to assume that $w_i \bullet x_k \neq w_i \bullet x_{k'}$ for all $k \neq k'$ so my error must be in there because this is the strongest assumption in the paper. From the figures it is easy to see that after the initial training it never got any better after seeing more examples. If I had more time, I would diagnose my error and find datasets that fit my algorithm. I think the name "Extreme Learning Machine" fits well with regard to the time to learn. The biggest bottleneck is calculating the inverse and all the matrix operations in the RLS algorithm.

References

[1]Huang, Guang-Bin & Liang, Nanying & Rong, Hai-Jun & Saratchandran, Paramasivan & Sundararajan, Narasimhan. (2005). On-Line Sequential Extreme Learning Machine.. Proceedings of the IASTED International Conference on Computational Intelligence. 2005. 232-237.

[2]Huang, Guang-Bin, et al. "Extreme Learning Machine: Theory and Applications." *Neurocomputing*, vol. 70, no. 1–3, Dec. 2006, pp. 489–501. *EBSCOhost*, doi:10.1016/j.neucom.2005.12.126.

Individual Report: Jake Prusky

1. Overview:

I chose to investigate the Follow the Leader (FTL) algorithm. The algorithm is fairly simple, there are a bunch of experts that suggest strategies and we choose to follow the one that has the best reward or lowest cost. This approach is online because we track how each expert performs over all of the previous steps to find which experts perform the best over time.

The data I chose to study in addition to our collective data was a hearth failure prediction dataset from BMC Medical Informatics and Decision Making, and data regarding rain in Australia from Commonwealth of Australia, Bureau of Meteorology.

This algorithm is traditionally studied from an adversarial standpoint, which is valid because that is the worst case scenario for a problem. The paper "On minimaxity of Follow the Leader strategy in the stochastic setting" looks at this algorithm in all cases excluding the worst case for the simple reason that most real world scenarios are not purely adversarial. In this paper four metrics were studied: loss, expected regret, pseudo regret, and excess risk. This was an interesting paper that provided insight on bounds of loss and regret functions for certain scopes of the problem.

It is known that FTL works very well for problems in which the loss functions are convex and positively curved, but the paper "Following the Leader and Fast Rates in Online LinearPrediction: Curved Constraint Sets and Other Regularities" seeks to explore what other scenarios, if any, might FTL be a good fit. Interestingly, the writers showed that as long as the mean of the loss vectors have positive lengths, the algorithm will have logarithmic regret.

For my research extension I wondered if it were possible that our experts weren't as smart as we presumed them to be, or maybe they are too short sighted. I wanted to see if adding in a small random possibility of choosing an expert we thought was not the best may have positive implications down the line. To do this, I simply add a constraint *p* and *p* percent of the time we will choose a random expert instead of the one with the lowest loss. The goal of this is to possibly account for features that are not represented, or experts that may have overfit.

2. Algorithm

T - the number of trials
K - the number of experts each with a distribution over loss values P_K
prediction weights $w_t = (w_{t,1}, w_{t,2}, \ldots, w_{t,K})$
the loss vector l is a function of P_K

the cumulative loss of an expert k at the end of round t, $L_{t,k}$ =

$$\sum_{q<t} l_{q,k}$$

to minimize loss we want to find

$$\sum_{t=1}^{T} w_t \cdot l_t - min_k L_{T,k}$$

3. Research Extension

To add in a factor of randomness have another hyperparameter λ between 0 and 1 that will be the probability of not listening to the expert's advice

Let X be a randomly generated number between 0 and 1

Now instead of

$$\sum_{t=1}^{T} w_t \cdot l_t - min_k L_{T,k}$$

we have

if X < λ

$$\sum_{t=1}^{T} w_t \cdot l_t - random(k)$$

else

$$\sum_{t=1}^{T} w_t \cdot l_t - min_k L_{T,k}$$

4. Discussion

When I first encountered FTL I immediately saw similarities to Q-Learning which as it turns out is the basis of some types of FTL variants. The difference between Follow the Regularized leader and Follow the Moving Leader encapsulates this. Normal FTL does not account for recency of changes in loss functions and is therefore not as optimized. I had trouble translating the data from the loss of information to actual predictions so I was not able to find the accuracy of my predictions.

It did not seem that the research extension I provided had any positive impact on the loss or accuracy for the datasets that I used, but I do not think that it necessarily means it is not useful in any case.

References:

Kotłowski, Wojciech. "On Minimaxity of Follow the Leader Strategy in the Stochastic Setting." *Lecture Notes in Computer Science*, 2016, pp. 261–275., doi:10.1007/978-3-319-46379-7_18.

Ruitong, Huang. et al. "Following the Leader and Fast Rates in Linear Prediction: Curved Constraint Sets and Other Regularities" arXiv:1702.03040

Shuai, Zheng, James T. , Kwok “Follow the Moving Leader in Deep Learning”

Individual Report: Michael O’Brien

1. Overview / Papers / Datasets

The algorithm that I implemented was LASVM, which is a support vector machine algorithm designed to handle a steady stream of online data. Traditionally support vector machines learn via batch data, or data that is already all known, LASVM is able to converge to the traditional SVM solutions given these small batches of data one at a time while performing a single pass over the entire training set. The LASVM algorithm will use less memory and will converge to a solution significantly faster than traditional SVM solvers given a stream of online data.

When looking to solve this online support vector machine problem, I first wanted to find a paper which detailed in depth the LASVM problem. “Fast Kernel Classifiers with Online and Active Learning” details the LASVM algorithm in which I attempted to implement. After, I wanted a feasible and reliable solution to solving quadratic programming problems. Thus I found a paper, “Quadratic Programming with Python and CVXOPT” , which uses the CVXOPT package detailing a relatively simple solution to these complex problems. Lastly, for my research extension I wanted to explore the effects of non traditional kernels and a soft margin within support vector machines and how they can be used to better classify data, thus finding the paper “Classes of Kernels for Machine Learning: A Statistics Perspective”.

As a group, we had our two joint evaluation dataset, “vec_loan” and “vec_spam” which are described below. For my individual datasets, I used two “orange_grapefruit.csv” and “cardio_disease.csv”. The orange vs grapefruit dataset is composed of 5 feature vectors (diameter, weight, red, green, blue) and lastly, there is a label vector at the end with class labels (0 , 1) with a 0 representing an orange and a 1 representing a grapefruit. Training on his dataset will be to tell the difference between grapefruits and oranges. The other dataset I used was “cardio_disease.csv” which has 11 feature vectors (Age, Height, Weight, … etc) with class labels (0 , 1) with 0 representing absence of cardiovascular disease and 1 representing the presence of cardiovascular disease. Training on this dataset will allow the algorithm to tell whether there is the presence or absence of cardiovascular disease.

2. Algorithms / How they were implemented / Libraries

The first algorithm that I attempted to implement was LASVM. This algorithm is built off of the idea of support vector machines. In general, support vector machines want to draw a hyperplane or margin that can satisfy the equation:

$$w^T \bullet x - b = 0$$

where w is the weight vector and x is the feature vector. Support vector machines do this by mapping out feature vectors through kernels to draw hyperplanes such that they can be classified into binary bins. LASVM builds upon this idea. It aims to satisfy this dual convex optimization problem:

$$W(\alpha) = \sum_i \alpha_i y_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)$$

We want to maximize this function with respect to α:

$$max_\alpha W(\alpha)$$

with constraints,

$$\sum_i \alpha_i = 0$$
$$A_i \leq \alpha_i \leq B_i$$
$$A_i = min(o, Cy_i) \;\&\; B_i = max(0, Cy_i)$$

where α_i is the coefficients of the SVM kernel expansion and $K(x_i, x_j)$ uses the kernel trick to save memory. Instead of storing all the mapped values, we can just calculate the kernel function in order to optimize our problem, thus saving a lot of space in our program. LASVM continuously solves this quadratic programming problem using online data. It does so by checking first whether or not the new data already satisfies our current model of the system. If it does, then nothing happens. If the new data passed to our system isn't satisfied by our model, it solves the quadratic programming problem with the new data stored in our training data. This works because if any new data is already satisfied by my model, then retraining the model would only yield the same results as the margin wouldn't change. LASVM can be summarized as follows:

1) initiazliation:
    a) Pass to it the initial samples to the train function
    b) set alpha ← 0 and compute the initial gradient g (quadratic program problem)
2) Online iterations:
    a) Repeat a predefined number of times:
        i) pick an example k
        ii) if k does not fit within model
            (1) Run update(k)
        iii) Run the preprocess once
3) Finishing
    a) Repeat Reprocess until quadratic problem is properly solved with k now a part of batch data

When looking to solve this quadratic programming problem, I found the CVXOPT library. This library will solve quadratic programming problems of the form:

$$min_x \frac{1}{2}x^T Px + q^T x$$
$$\text{subject to } Gx \leq h$$
$$Ax = b$$

A, b, P, q, G, h are all input matrices to help solve the equation. This was extremely useful for optimizing and saving space within the code as it efficiently and reliably solves these quadratic programming problems. Lastly, in order to split the data into testing and training data, I used the train_test_split function in the sklearn package. The other libraries were all the basic one, numpy, pandas, math, etc

3. Research Extensions

For my research extension, I aim to explore 5 different kernel functions other than the standard linear kernel.

polynomial:

$$K(x_1, x_2, p) = (1 + x_1 \bullet x_2)^p$$

rational quadratic:

$$K(x_1, x_2, \theta) = 1 - \frac{||x1-x2||^2}{||x1-x2||^2+\theta}$$

exponential:

$$K(x_1, x_2, \theta) = exp(-\frac{||x1-x2||}{\theta})$$

gaussian:

$$K(x_1, x_2, \theta) = exp(-\frac{||x1-x2||^2}{\theta})$$

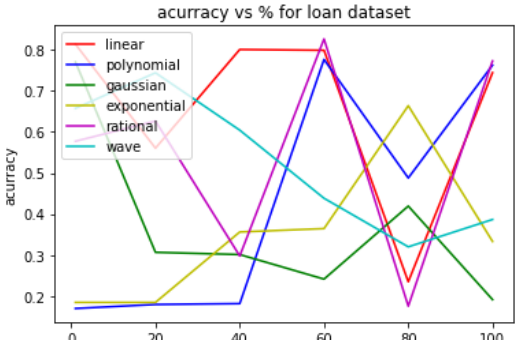
wave:

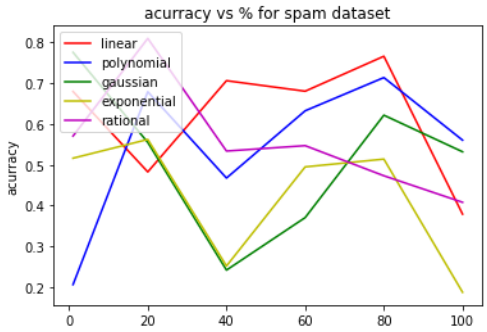
$$K(x_1, x_2, \theta) = \frac{\theta}{||x1-x2||} sin(\frac{||x1-x2||}{\theta})$$

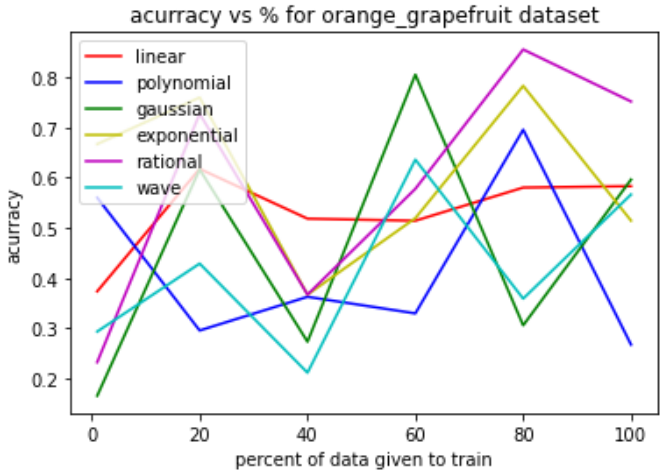
I will test all of these kernel functions against my linear kernel function to see how they compare in performance.

4. Experiments / Results

All of the code to obtain these results is within the svm_notebook.ipynb file on the repository. I fed in the training data in batch sizes of 20% of its total size so that we could see the effects of the online functionality of my support vector machine. Below are graphs comparing the linear kernel function with the other kernel functions for each dataset. The graphs plot accuracy vs percent of training data available to the program (%).







5. Conclusion / Discussion

Based off the results in the experiments / results section, it is clear that my algorithm was inconsistent. While there does seem to be a lot of noise within the online learning functionality, within some of the datasets, you can see some trends. For example within the orange vs grapefruit dataset, the rational kernel performs better at every level over the polynomial kernel. While in the spam dataset, the rational kernel generally performed worse than the polynomial kernel. This leads me to the conclusion that kernels are better for specific datasets. One kernel may yield amazing results for one dataset, while the other may do no better than chance.

Another conclusion drawn from my experiments is that even though my data was generally inconsistent within accuracy, when I reran them over the same test data, the values stayed relatively constant. Meaning that it was creating some sort of classifier on the data rather than simply guessing. Whether or not that classifier yielded appropriate results, I cannot say, but I can say it is consistently getting the same accuracy.

6. Refrences

1) Bordes, Antoine, et al. “Fast Kernel Classifiers with Online and Active Learning.” *Journal of Machine Learning Research*, Sept. 2005.
2) Genton, Marc G. “Classes of Kernels for Machine Learning: A Statistics Perspective.” *Journal of Machine Learning Research 2*, Dec. 2001.

3) Subero, Armstrong. “Python Programming.” *Programming Microcontrollers with Python*, 2021, pp. 107–125., doi:10.1007/978-1-4842-7058-5_4.

Joint Evaluation

Datasets

1. SMS Spam dataset
    - SMS Spam Dataset contains the raw SMS text contents as the feature. The stop words and non-character words are removed from the raw data. Then the text contents are tokenized and vectorized using the nltk tool kit. A generated 2500-length feature list is used for learning.
2. Loan dataset
    - The feature values in the Loan Dataset are normalized before the training.

Sohan Bhawtankar:

Dataset	**Accuracy**
Spam Dataset	0.98
Loan Dataset	0.662

Zhou Yang:

RBP Algorithm:

Dataset	**Accuracy**
Spam Dataset	0.86
Loan Dataset	0.84

RRF Algorithm:

Dataset	**Accuracy**
Spam Dataset	0.88
Loan Dataset	0.86

Ellis Wright:

Dataset	**Accuracy**
Spam Dataset	N/A*****
Loan Dataset	0.83

*** When calculating the weights, the matrix is not invertible so there is no solution with my implementation.**

Michael O'Brien:

Dataset	**Accuracy**
Spam Dataset	.65
Loan Dataset	.74

Jake Prusky:

I was not able to produce a number for accuracy on the joint datasets

Dataset	**Accuracy**
Spam Dataset	N/A
Loan Dataset	N/A

Group Discussion

Looking at our group's accuracy for the loan data set, we have reason to believe that data collected is not the best representation of the learning problem. A classifier that predicts all negatives should be correct 83.83 % of the time. The highest any one of our algorithms had is 86%- roughly 2% better. Another possible reason is that the preprocessing done on the original dataset lost too much information.

Also, it appears that Sohan's algorithm does the best job in classifying the data in the spam dataset. We believe this is due to his algorithm being able to handle features that have binary values. The spam dataset is generated from raw text contents with filtering out the stopwords and then converted to one-hot encoding of 2500 words. We speculate that due to this nature of the dataset, some algorithms, like the Perceptron implemented by Sohan, might be more suitable than others. The datasets might be too sparse in feature space in this case. In addition, the parameters used for encoding/embedding the raw texts might also influence the representation of the dataset and then influence the evaluation results. With more testing, we believe this could extend to being able to classify features that have non-binary categorical values (more than two) better than the other algorithms.

By revisiting the nature of the algorithms we implemented and comparing the results to the pre-built Perceptron in sklearn package which also does not have superior performance on the dataset, we ruled out the probability that our implementations are flawed. And since for individual evaluations, our algorithms also have various performance among different datasets, we believe that the performance. Moreover, we conclude that most of our algorithms are linear algorithms. This might also be an explanation for their decay in performance when it comes to spam and loan datasets, those 2 datasets could be infeasible to have a linear or near-linear classification results or even, the features/examples we have in the datasets could not support to find such solution. We also observed similar situations in individual datasets.

From this perspective, combined with the papers we read, we can see that in this survey, most of the online kernel learning approaches for large-scale datasets are actually trying to find efficient methods to make the existing linear algorithms to work on the non-linear problems with large feature space or large sample space. However, some of the problems might not be feasible for this kind of conversion, like Ellis' learner could not process spam dataset and some of the problems could not be well solved/approximated by those methods. After all, we believe that the first priority of large-scale online learning is to reduce the resources needed to proceed the learning tasks into a reasonable, affordable level which makes large-scale learning feasible. The performance and the generality/compatibility is of course important but need to yield to feasibility.

Furthermore, for the 2 joint datasets we evaluated, there are some decision tree methods and deep learning methods working well with good performance. But there is no good modification or extension of them for large-scale online learning scenarios in our research scope. An approach to combine the efficiency of the online kernel learning and the affinity of the suitable batch learning methods will be valuable for solving this kind of problem.

References

1.Koby Crammer, Jaz S Kandola, and Yoram Singer. Online classification on a budget. In Neural Information Processing Systems, volume 2, page 5, 2003.

2.Lu, Jing, et al. "Large scale online kernel learning." Journal of Machine Learning Research 17.47 (2016): 1.

3.Giovanni Cavallanti, Nicol`o Cesa-Bianchi, and Claudio Gentile. Tracking the best hyperplane with a simple budget perceptron. Machine Learning, 69(2-3):143–167, 2007.

4.Zhuang Wang and Slobodan Vucetic. Online passive-aggressive algorithms on a budget. In International Conference on Artificial Intelligence and Statistics, pages 908–915, 2010.

5.Francesco Orabona, Joseph Keshet, and Barbara Caputo. The projectron: a bounded kernel-based perceptron. In Proceedings of the International Conference on Machine Learning, pages 720–727, 2008

6.Zhuang Wang, Koby Crammer, and Slobodan Vucetic. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. Journal of Machine Learning Research, 13:3103–3131, 2012b

7.Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. Proceedings of the VLDB Endowment, 5(8):716–727, 2012

8.Nguyen, Tu Dinh, et al. "Large-scale Online Kernel Learning with Random Feature Reparameterization." IJCAI. 2017.

9. Cavallanti, Giovanni, Nicolo Cesa-Bianchi, and Claudio Gentile. "Tracking the best hyperplane with a simple budget perceptron." *Machine Learning* 69.2 (2007): 143-167.

10.Li, Tie, Gang Kou, and Yi Peng. "Improving malicious URLs detection via feature engineering: Linear and nonlinear space transformation methods." Information Systems 91 (2020): 101494.

11. Rong Jin, Steven C.H. Hoi, and Tianbao Yang. “Online Multiple Kernel Learning: Algorithms and Mistake Bounds”.

12. Huang, Guang-Bin & Liang, Nanying & Rong, Hai-Jun & Saratchandran, Paramasivan & Sundararajan, Narasimhan. (2005). On-Line Sequential Extreme Learning Machine.. Proceedings of the IASTED International Conference on Computational Intelligence. 2005. 232-237.

13. Huang, Guang-Bin, et al. “Extreme Learning Machine: Theory and Applications.” *Neurocomputing*, vol. 70, no. 1–3, Dec. 2006, pp. 489–501. *EBSCOhost*, doi:10.1016/j.neucom.2005.12.126.

14. Bordes, Antoine, et al. “Fast Kernel Classifiers with Online and Active Learning.” *Journal of Machine Learning Research*, Sept. 2005.

15. Genton, Marc G. “Classes of Kernels for Machine Learning: A Statistics Perspective.” *Journal of Machine Learning Research 2*, Dec. 2001.

16. Subero, Armstrong. “Python Programming.” *Programming Microcontrollers with Python*, 2021, pp. 107–125., doi:10.1007/978-1-4842-7058-5_4.