Kernelized Linear Classification Report

Miad Alavinezhad

September 2024

Professor: Cesa-Bianchi

Note: *I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.*

1 Introduction

In the following report, the performance of several classification algorithms will be stated and compared to each other. In the first part the following algorithms are discussed:

1. Perceptron
2. support Vector Machines (SVMs) using the Pegasos algorithm
3. Regularized logistic classification (Pegasos objective function with logistic loss)

In second part the performance of the above algorithms are compared using polynomial feature expansion of degree two. In the final part algorithms are compared while using kernel methods. These algorithms are:

1. kernelized Perceptron with the Gaussian and the polynomial kernels
2. kernelized Pegasos with the Gaussian and the polynomial kernels for SVM

Different steps from data preprocessing to training the models and evaluating them is explicitly documented before diving into comparing the methods and their performances.

2 Dataset and Methodology

The dataset consists of 10,000 samples, each with 10 features $x_1, x_2, ..., x_{10}$ and labels $y \in \{0, 1\}$. To prepare the dataset, the training set and test set are separated with the ratio of 85/15 (training set/test set). Usually the training set takes 70% to 80% of all the samples, but in this case as the number of samples are not very large, more data is assigned for training and evaluating while keeping a reasonable amount of samples for testing.

Since k-fold cross validation is being used with $k = 5$ for model evaluation, the indices for different folds are calculated. The k folds are made out of training set only($k - 1$ fold for training set, 1 fold for validation set) and **not the test set**.

For hyper-parameter tuning, Bayesian Optimization with Guassian Process is utilized, as evaluating models with different hyper-parameters can take a very long time. The optimization uses a surrogate model(in our case Guassian Process) to approximate the output of the objective function that needs to be optimized. Guassian Process(GP), models the objective function as a distribution over possible functions and updates as more data is gathered. The inputs of our objective function are hyper-parameters and the output is the average of the k-fold accuracy results. Next the acquisition function decides where to evaluate the objective function next and the GP is updated with new evaluation results. This process repeats until a certain amount of iterations and the final(best) hyper-parameters and accuracy is given to us.

Inside the objective function, an instance of a model is made with the hyper-parameters that the Bayesian Optimization gives to the function. Then inside the loop of k-fold cross validation, training and validation set are defined. The sets are then scaled using z-score scaling method since different features in the dataset have different ranges, scaling the data causes the features to have same importance to the algorithm and prevent some features to dominate others. See Figure 1. The model is then trained with scaled training set and evaluated with scaled validation set. The loop iterates for k iterations and the average of all those accuracies is returned.

Now that we have the optimal hyper-parameters, the model is once again trained with them and the scaled training set(85% of whole dataset) as input. The final evaluation is done with test set(scaled) which the model has never seen until this point.

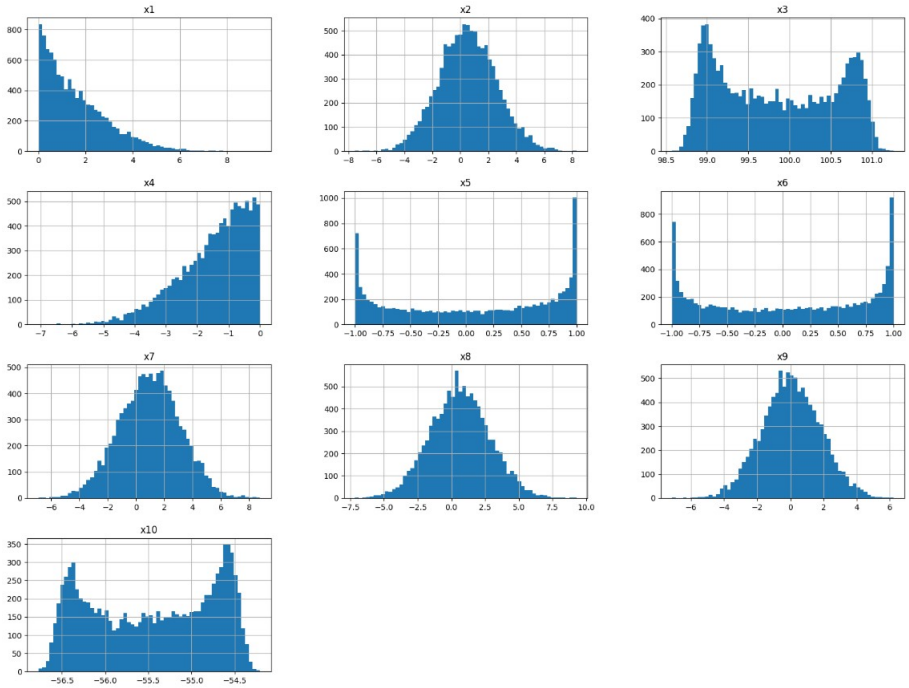

Figure 1: Histogram of dataset features - *Features x_3 and x_5 have very different ranges from other features and themselves*

3 Classification Algorithms

3.1 Perceptron

The Perceptron algorithm find a linear boundary (hyperplane) that separates two classes in a dataset. The Perceptron pseudo-code is as follows:

Algorithm 1 Perceptron

Parameters: T number of rounds, η learning rate
Data: $(x_1, y_1), ..., (x_m, y_m)$
w**:** $(0, ..., 0)$
for $t = 1, 2, ..., T$ **do**
  for $i = 1, 2, ..., m$ **do**
    if $y_i w^T x_i < 0$ **then**
      $w \leftarrow w + \eta * y_i x_i$
    end if
  end for
end for
output: w

For computational cost we consider number of iterations (T), number of features (n) and number of training samples (m). For each sample there is inner product with cost of $O(n)$ and in case of miss-classification another $O(n)$ is added. Therefore for each iteration the cost is $O(m \cdot n)$. There are T iterations, so the overall cost is $O(T \cdot m \cdot n)$

The results below is computed by B.O.(Bayesian Optimization) process on Perceptron:

H-Parameters	**Range**	**Optimal Value Found**
T	$200 - 1000$	620
η	$0.01 - 0.1$	0.012

Table 1: Perceptron H-Parameter Tuning

The average accuracy computed with k-fold in the B.O. process is 63.88 %. The **final accuracy** after training the model with given hyper-parameters and testing it with test set is **64.13 %**.

Accuracy of cross validation and final testing are close but since this classification is binary, the accuracy is fairly low so it can be a sign of **underfitting**.

3.2 SVM with Pegasos

Support Vector Machine(SVM) is a learning algorithm commonly used for classification and regression tasks. The goal of SVM is to find a hyperplane that best separates data points in different classes with the maximum margin possible. Pegasos is a stochastic gradient descent (SGD) based algorithm which is used to solve the SVM optimization problem. The soft-SVM objective is as follows:

$$\begin{aligned} min_{(w,\xi)} \quad & \frac{\lambda}{2} \parallel w \parallel^2 + \frac{1}{m} \sum_{t=1}^{m} \xi_t \\ s.t. \quad & y_t w^T x_t \geq 1 - \xi_t \\ & \xi_t \geq 0, \quad t = 1, ..., m \end{aligned} \tag{1}$$

The reason for using soft-SVM is that it is highly unlikely that the dataset be linearly separable. By using soft-SVM we are relaxing the constraint.

Algorithm 2 SMV Pegasos

Parameters: T number of rounds, λ regularization coefficient
Data: $(x_1, y_1), ..., (x_m, y_m)$
w: $(0, ..., 0)$
for $t = 1, 2, ..., T$ **do**
  Uniformly at random select a set (x_{z_t}, y_{z_t}) from training set
  if $y_{z_t} w^T x_{z_t} < 1$ **then**
    $w \leftarrow (1 - 1/t)w + \eta * y_{z_t} x_{z_t}$
  else
    $w \leftarrow (1 - 1/t)w$
  end if
end for
output: $\bar{w} = \frac{1}{T}(w_1, ..., w_T)$

Computation cost for this algorithm is $O(T \cdot n)$. In each iteration (which there are T in total), at least one inner product between the sample and corresponding weight is performed which the cost is $O(n)$

The results below is computed by B.O.(Bayesian Optimization) process on SVM Pegasos:

H-Parameters	**Range**	**Optimal Value Found**
T	10,000-100,000	99806
λ	$0.01 - 0.5$	0.011

Table 2: SVM Pegasos H-Parameter Tuning

The average accuracy computed with k-fold in the B.O. process is 73.01 %. The **final accuracy** after training the model with given hyper-parameters and testing it with test set is **71.33 %**

Results of cross validation and final testing is close so there is no overfitting. The accuracy numbers does not show an underfitting situation, rather it shows relatively good generalization but can still be better. Therefore **no overfitting or underfitting**

3.3 Pegasos Algorithm with Logistic Loss

For a Regularized Logistic Classification, *Pegasos Algorithm with Logistic Loss* is considered. The algorithm is basically the Pegasos algorithm which hinge loss is replaced by logistic loss:

$$f(w) = \frac{\lambda}{2} \parallel w \parallel^2 + \frac{1}{n} \sum_{i=1}^{m} log(1 + e^{-y_i w^T x_i}) \tag{2}$$

The pseudo code of the algorithm can be seen as below:

Algorithm 3 Pegasos Logistic Loss

Parameters: T number of rounds, λ regularization coefficient
Data: $(x_1, y_1), ..., (x_m, y_m)$
w: $(0, ..., 0)$
for $t = 1, 2, ..., T$ **do**
  Uniformly at random select a set (x_{z_t}, y_{z_t}) from training set
  if $y_{z_t} \neq \sigma(w^T x_{x_t})$ **then**
    $w \leftarrow (1 - 1/t)w + \eta(\sigma(w^T x_{x_t}) - y_{z_t})x_{z_t}$
  end if
end for
output: w

Computational cost for the above algorithm is computed by considering the number of iteration(T) and number of features(n) since in each iteration, a inner product is performed with cost of $O(n)$. The overall computational cost is $O(T \cdot n)$

The results below is computed by B.O.(Bayesian Optimization) process on Pegasos with logistic loss:

H-Parameters	**Range**	**Optimal Value Found**
T	10,000-100,000	68580
λ	$0.01 - 0.5$	0.08

Table 3: Pegasos with logistic loss H-Parameter Tuning

The average accuracy computed with k-fold in the B.O. process is 72.51 %. The **final accuracy** after training the model with given hyper-parameters and testing it with test set is **71.2 %**

Situation in this case is very similar to SVM, close accuracy between cross validation and final test, and good generalization, so in this case there's **no sign of underfitting and overfitting**.

4 Feature Polynomial Expansion

Next part of the project the models are trained on dataset with polynomial features of degree two. Features of a sample in the dataset before expansion are $X = \{x_1, ..., x_{10}\}$ and after polynomial expansion, it has all original features plus all squared features and all pairwise multiplication between them. Number of new features created is calculated by:

$$\frac{n(n+1)}{2}$$

where n is number of original features. In our case the number of original features are 10 which gives us 65 total features after expansion:

$$X_{polynomial} = \{x_1, ..., x_{10}, x_1^2, ..., x_{10}^2, x_1x_2, x_1x_3, ..., x_9x_8, x_9x_{10}\}$$

Now the comparison between the accuracy of models before and after feature expansion:

Model	**Regular Features**	**Expanded Features**	**Improvement**
Perceptron	64.13 %	88.6 %	+24.47 %
SVM Pegasos	71.33 %	91.13 %	+19.8 %
Pegasos Logistic	71.2 %	79.46 %	+8.26 %

Table 4: Accuracy comparison

A clear improvement after feature expansion, which means the new features help the models to capture more complex patterns.

The comparison between the original weights and linear corresponding weights of the various numerical features after training the model are as follows:

Model	**AVG Original**	**AVG Linear(Expanded)**	**Absolute Diff.**
Perceptron	0.0035	-0.0010	0.0045
SVM Pegasos	0.1996	0.0413	0.1582
Pegasos Logistic	2.0422	0.2651	1.7770

Table 5: Weight comparison of original features and linear features of polynomial expansion

When using expanded features, the computational cost also changes for the methods as the number of features have quadratic growth. Therefore whenever $O(n)$ is used, its needs to be replaced by $O(n^2)$:

- Perceptron: $O(T \cdot n^2 \cdot m)$
- SVM Pegasos: $O(T \cdot n^2)$
- Pegasos Logistic: $O(T \cdot n^2)$

Cross validation for Perceptron, SVM Pegasos and Pegasos Logistic are 88.5%, 90.68% and 79.67% respectively which are close to the final test accuracy. Also all have high percentages of accuracy. In none of the algorithms there is **no sign of underfitting or overfitting**.

5 Kernel Methods

In the final part, the kernel methods are used to map data into higher dimensions of feature space in order to find a better linear separation. Kernel methods transform the original input space into a higher-dimensional space without computing the feature transformation, using kernel trick.

In this case Perceptron and SVM algorithms use both Polynomial and Guassian kernels. For polynomial kernel, the degree in which the input data is going to be mapped needs to be tuned. The higher the degree, the more complex the decision boundary. If the decision boundary gets to complex, the model can experience overfitting for learning the training set too well.

The same can happen for Guassian kernel by adjusting the σ. When σ is small the Gaussian kernel becomes sensitive to the distance between data points. Only points that are close to each other in the feature space will make a large kernel values which can lead to overfitting.

Polynomial Kernel:

$$k(x, z) = (x^T z + c)^d$$

Guassian Kernel:

$$k(x, z) = exp(-\frac{(\| x - z \|^2)}{2\sigma^2})$$

5.1 kernelized Perceptron

Perceptron algorithm using kernel methods:

Algorithm 4 kernelized Perceptron

Data: $(x_1, y_1), ..., (x_m, y_m)$
for $t = 1, 2, ..., m$ **do**
  $\hat{y}_t = sgn(\sum_{s \in S} y_s k(x_s, x_t))$
  if $y_t \neq \hat{y}_t$ **then**
    add t to S
  end if
end for
output: S

Computational cost for kernelized Perceptron can be calculated by considering number of iterations(T) and the computation in each epoch with is computing kernel function. Kernel function is polynomial kernel consists of inner product between two samples with n features with cost of $O(n)$ and for Guassian kernel, subtracting one sample from another with n features with cost of $O(n)$. So the

overall cost for kernelized Perceptron with either polynomial or Guassian kernel is $O(T \cdot n)$

Results of Perceptron algorithm on kernel methods after hyperparameter tuning and final testing:

Kernel	**Optimal Degree**	**Optimal Sigma**
Polynomial	4	-
Guassian	-	0.79

Table 6: Accuracy of different kernels with Perceptron

Model's performance in cross validation on Polynomial kernel is **91.43 %** and on Guassian kernel is **90 %**. The **final test accuracy** for Polynomial kernel is **88 %** and **91.93 %** for Guassian kernel.

There is no underfitting or overfitting since the cross validation results are close to final test accuracy which both are high.

5.2 kernelized Pegasos for SVM

SVM algorithm using kernel methods:

Algorithm 5 kernelized Pegasos for SVM

Parameters: T number of rounds
Data: $(x_1, y_1), ..., (x_m, y_m)$
for $t = 1, 2, ..., T$ **do**
  Uniformly at random select a set (x_{z_t}, y_{z_t}) from training set
  $\hat{y_t} = sgn(\sum_{s \in S} y_s k(x_s, x_t))$
  if $y_t \neq \hat{y_t}$ **then**
    add t to S
  end if
end for
output: S

Computational cost of kernelized Pegasos for SVM is computed as same as kernelized Perceptron. cost of kernels are $O(n)$ which are in a loop of T iterations. Overall computation is $O(T \cdot n)$.

Results of SVM with Pegasos algorithm on kernel methods:

Kernel	Optimal Degree	Optimal Sigma	Iterations
Polynomial	2	-	6000
Guassian	-	0.99	5340

Table 7: Accuracy of different kernels with SVM

Model's performance in cross validation on Polynomial kernel is **90.9 %** and on Guassian kernel is **89.4 %**. The **final test accuracy** for Polynomial kernel is **90.86 %** and **88 %** for Guassian kernel.

There is no underfitting or overfitting since the cross validation results are close to final test accuracy and both have are high values.

6 Conclusion

Algorithms discussed in this report are linear classification methods, used to find a linear hyperplane. The given dataset has too many features and complexity to be linearly separable therefore these algorithms did not perform very well on their own. As the result, feature expansion and kernel methods are used so that these methods can capture the more complex relations between samples and shape more complex boundaries to fit the data and as the results show, a significant improvement has been made using them.