Image Classification

Jun Lu
EPFL
jun.lu@epfl.ch

Abstract

In this report, we summarize our finding regarding Image Classification. Our goal was to design a system which recognizes whether a certain object is present in a given image. Two labeled sets of input features were provided. This is a supervised learning classification problem whose performance is measured by the Balanced Error Rate. Both binary predictions (one versus rest) and multi-class predictions were to be made. We found out that CNN features alone conducted to significantly better results. These features were trained according to supervised learning ML methods such as Logistic Regression, SVM's and Neural Networks. Our best predictions came from the application of Multiclass SVM's for both Binary and Multiclass Problems. We estime a Test Error **0.0825 (with standart deviation of 0.0077)**; for Binary, it is **0.0875 (with standart deviation of 0.069)**. Reducing the problem's dimensionality, dealing with imbalanced datasets and outlier removal through unsupervised learning were crucial to increase performance.

1 Feature Extraction

1.1 Data Description and Interpretation

Our data set consist of two sets of input features, each one of them split into training dataset and testing dataset.

The training dataset consist of $\mathbf{N_{tr}}$=6000 data samples of one output variable y and input variable $\mathbf{X}$. The input variable $\mathbf{X}$ depends on the feature provided, as further detailed underneath. The output variable is present is a 1-K coding scheme. The number of classes is K=4, having the class 1 **964** data samples; the class 2 **1162** data samples; the class 3 **1492** data samples the class 4 **2382** data samples.

The testing dataset consist of $\mathbf{N_{te}}$=11453 , for which the output is unknown, and where we forged our predictions.

1.1.1 HOG Features

The number of input variables for *HOG* features is $\mathbf{D}$=5408. *HOG* features result of decomposing an image into small squared cells, computing the histogram of oriented gradients in each cell, normalizes the result using a block-wise pattern and then returning the descriptor for each cell. Each dimension of the input variable consist of a histogram bin reveling the number of descriptors having a certain orientation. From this analysis, one can infer that *HOG* features are sensible to scaling and translation of the aimed object.

1.1.2 CNN Features

The number of input variables for *CNN* features is $\mathbf{D}$=36865. *CNN* features consist of training an image over a convolutional network. For the images provided, in particular, the network consisted of 5 convolutional layers and 2 fully connected layers. Each convolutional layer gathers information about small portions of the previous layers and posteriorly combining them. This away, it is possible to tolerate of the translation of the aimed object.

1.2 Feature Engineering

In the first place, due to observations made concerning the input variables distribution, we standardized the data set through standard score. In the second place, we concluded high dimensionality of the problem conveys issues such as very high computation cost, in terms of memory and running time, and sparsity of the data - low data density and consequently worse statistical results. Therefore, we applied **PCA** factorization to the input feature through **SVD** decomposition, selecting the first **50 principal components**. [1] Furthermore, the resulting input feature as a normal distribution over the input space, which is a desirable requirement as the probabilistic models behind of the algorithms applied are driven from a normal distribution.

1.3 Identification of Outliers

We addressed the question of outlier removal making use of unsupervised learning methods, in particular through the implementation of **K-Means algorithm**. No external code libraries were used in the implementation of this learning Algorithm. Our approach was based on the fact that each characteristic class (either class 1, 2 or 3) should in theory be restricted to a certain volume of the input space. Bearing this fact in mind, each of these three classes can be thought as a cluster. Consequently, we consider a sample to be an outlier if it does not belong to its labeled class cluster.

Having said that, we applied K-Means to a dataset consisting of only the samples of classes 1, 2 and 3. In the case of *CNN* features, 262 were tagged as outliers. In the case of *HOG* features, 641 were tagged as outliers. The choice of the initial conditions was crucial in terms of the algorithm's convergence and in terms of obtaining a reliable result. [2]

1.4 Solving Imbalanced Data Set

Concerning a classification problem, an imbalanced data set leads to biased decisions towards the majority class and therefore an increase in the generalization error. As referred in the **(1.1)**, the number of samples per class in our problem is not equally distributed. To address this problem, we considered three approaches.

- **Random Under Sampling** - Randomly select a subset the majority classes' data points so that the number of data points of each class is equal to the number of points of the minority class - Class 1.
- **Random Over Sampling** - Randomly add redundancy to the data set by duplicating data points of the minority classes so that the number of data points of each class is equal to the number of points of the majority class - Class 4.
- **Algorithmic Over Sampling** - Add redundancy to the data set by simulating the distribution of each minority class and consequently creating new data points so that the number of data points of each class is equal to the number of points of the majority class - Class 4.[3]

Further considerations would be applying our algorithms such that different classes were penalized differently (having a different regularizer to each class, proportional to its size); change the class label in the SVM model to -1/(K-1) instead of -1. These were not implemented.

[1] See Implementation Details - 4.1 PCA factorization
[2] See Implementation Details - 4.2 K-Means
[3] See Implementation Details - 4.3 ADASYN

2 Methods for Classification

2.1 Methodology

Firstly, we analyzed the definition of our performance index - Balanced Error Rate (BER). The Balanced Error Rate equally penalizes a misclassification, regardless of the size of the class. This is the main reason behind our imbalanced dataset approach. Secondly, we considered how to tackle the fact that apart from multiclass predictions (to which class does a sample belong to), a binary prediction (whether a sample belongs to Class 4) was also to be made. Towards that goal, we applied our methods through both perspectives: binary classification and multiclass classification, picking the one which produced the best result for each case. Thirdly, we dealt in the issue of having two sets of input features. We addressed three cases: having *HOG* features alone; having *CNN* features alone; having *HOG* and *CNN* features combined. According to observations made in **(1.1)**, and has proved in practice, *CNN* feature alone induced the best result. Hence, further on, we will only refer to them.

We trained and tested our dataset under three methods: **Penalized** and **not Penalized Logistic Regression** with different basis functions; **Support Vector Machines** with different kernels; **Neural Networks**, and always analyzing the performance of our approach towards solving imbalanced data sets. We set our baseline to be a random decision for the classification of the input, which leads to **BER=0.75**. We picked the best model as being the one which had the lowest **Cross Validation Error**.

Finally, we computed a prediction for the test error and forged predictions in the given test set. In the first case, we regarded 25 percent as Validation Set and 75 percent and Training Set. We performed feature engineering over the Validation Set and Training Set together ((see **(1.3)**) and posteriorly completed the feature extraction over the Training Set (see **(1.4 and 1.5)**). We trained our best model over the Training Set and then computed predictions for the Validation Set. The reliability of the predictions are conditioned by the fact that PCA was performed jointly for the Validation Set and the Training Set and that there is a sharp difference between the size of our Validation Set and the size of the Test Set. In the second case, we computed the parameters given by what we considered to be the best model by training over the whole original Training Set, after complete feature extraction; posteriorly, we projected the Test Set over the feature space of the Training Set and estimated its labels through the referred parameters.

2.2 Logistic Regression

A linear model for classification, such as Logistic Regression, was believed to be the simplest choice for a model. In terms of its implementation, caution was need due to the large size of the data. Instead of gradient descent, we implemented stochastic gradient descent together with a fading learning rate, which must be adapted to each dataset in particular, according to its size and range. [4]. Both polynomial and Gaussian basis functions were tried. None of them improved results when compared with the original linear basis. Consequently, we will not refer to them further on. From this fact we can deduce that the boundaries between each class are approximately linear and therefore a linear model provides a reasonable fit. No external code libraries were used in the implementation of this learning Algorithm.

For the multiclass classification problem, we considered **two approaches**: **One vs Rest** approach; **Multinomial Logistic Regression**. For the binary classification problem, we considered **three approaches**: the two mentioned above together with Logistic Regression classification of whether a sample belongs to class 4.

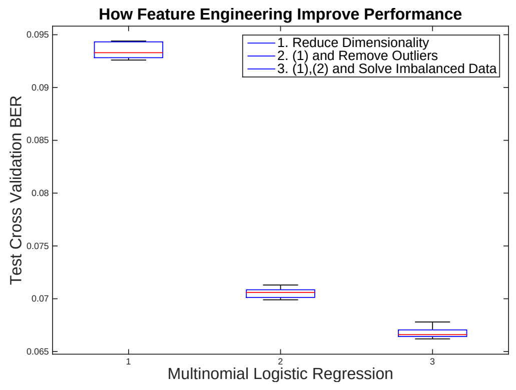
The **One vs Rest** approach consists of consecutively solving a two-class problem of whether a sample belongs or not to the class K. When recurring to Logistic Regression to solve this binary classification problem, the posterior probability $P(C_k|\theta)$ is obtained as output. Hence, by labeling members of the class K with 1 and the rest with 0, the decision rule for each sample consist of picking the class which maximizes the posterior probability, i.e.:

K=$max_k P(C_k|\theta)$

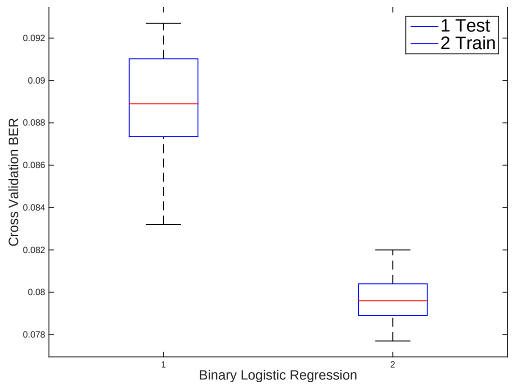
[4]See Implementation Details - 4.4 Stochastic Gradient Descent and Learning Rate

The **Multinomial Logistic Regression** approach consist of a generalization of the binary logistic regression. See **Appendix A** for the derivation of the Multinomial Logistic Regression.

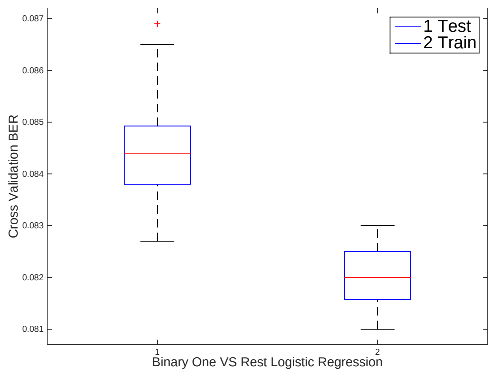
The models were trained using 4-Fold Cross Validation and ran over 25 randomly picked seeds, regardless of the feature engineering conducted over the input variable. A summary of how the feature engineering performed successively improve the Cross Validation error is presented in Figure 1(a). Furthermore, the performance of the various approaches is present in Figure 1(b) 1(c) 1(d) 1(e) 1(f). We can therefore conclude that in both cases the Multinomial approach conveys the best results, achieving a **Test Cross Validation Error of 0.0812 and 0.0626** for the binary and multiclass problems, respectively.



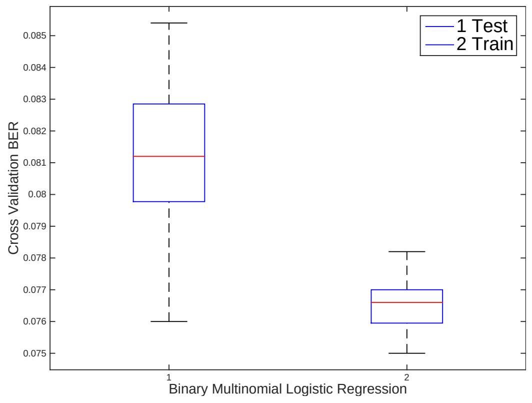
(a) How Feature Engineering Improve Performance.



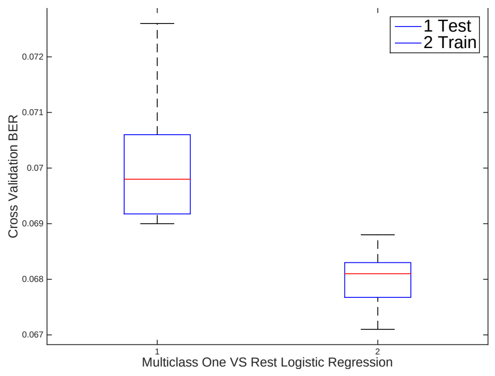
(b) Binary Logistic Regression.



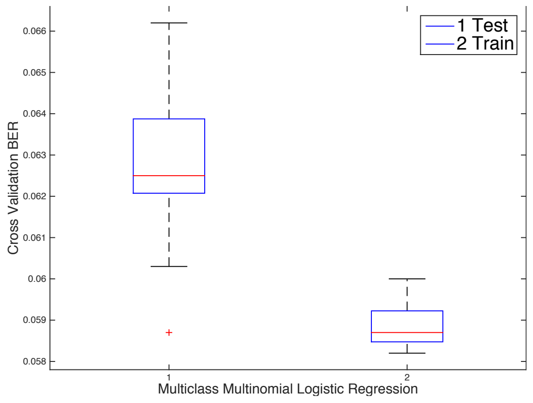
(c) Binary One VS Rest Logistic Regression.



(d) Binary Multinomial Logistic Regression.



(e) Multiclass One VS Rest Logistic Regression.



(f) Multiclass Multinomial Logistic Regression.

Figure 1:

2.3 Support Vector Machines

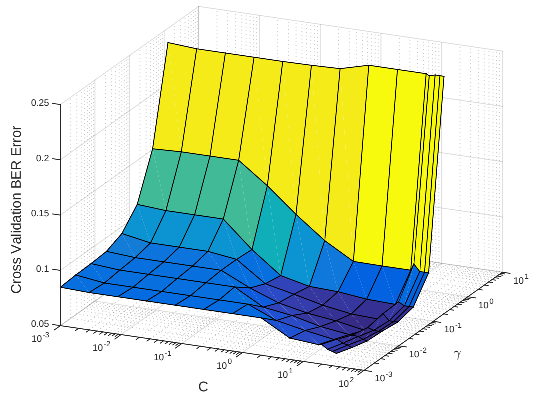
Following considering a linear model for classification, a maximum margin classifier, such as the Support Vector Machines, was fit. Our choice for the Kernel was towards the **Linear Kernel**, **Polynomial Kernel** and **Radius Basis Function (RBF) Kernel**. No external code libraries were used in the implementation of this learning Algorithm.

For the multiclass problem, we consider the approach **One vs Rest**, described in **(2.2)**. Even though SVM's do not have a straightforward probabilistic interpretation, we can associate its output with the probability of the machine making a correct decision. Hence, by labeling members of the class K with 1 and the rest with -1, the decision rule for each sample consist of picking the class which maximizes the probability of a correct decision:

$C_k = max_k \; \hat{y_k}$

For the binary class problem, the above approach was considered, together with binary SVM classification of whether a sample belongs to class 4.

For a matter of computational cost, but accounting its lower performance, we considered the regularized C to be equal for every class. Its optimization, together with the one of the other hyper parameters, was achieved through successive insightful grid search. A heuristic approximation was considered to have a first-hand insight on the value of the inverse variance, γ, in the case of the RBF Kernel. The BER surface as function of both C and γ, for the case of the RBF Kernel, is presented in Figure 2(a). The hyper parameter C represents a regularizer stablishing the cost of a misclassification. A small value of C conveys a low penalization for a misclassification, which implies that only few data points are taken as support vectors. On the other hand, a large value of C conveys a high penalization for a misclassification, which implies that a large number of data points are taken as support vectors. The hyper parameter γ specifies the range of influence of the each support vector. Consequently, a large γ issues a narrow sphere of influence of each support vector, leading to possible over fitting, while a small γ conveys a broader sphere of influence of each support vector, leading to possible under fitting. In sum, the optimal hyper parameters define a bias-variance tradeoff between the sphere of influence of each support vector and the number of support vectors selected.

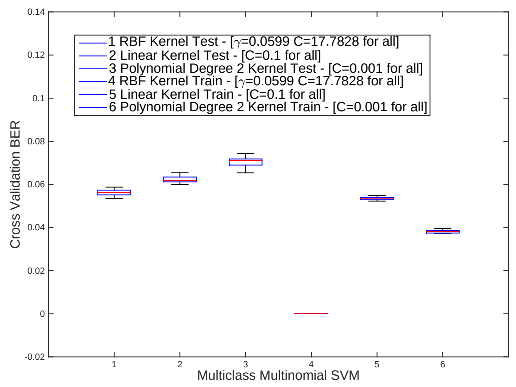


(a) Error Surface for Multiclass SVM with RBF Kernel.

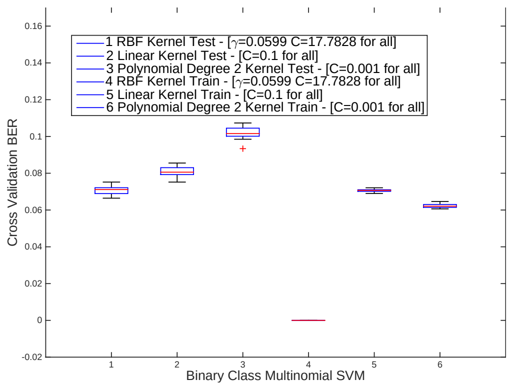
Figure 2:

The models were trained using 4-Fold Cross Validation and ran over 25 randomly picked seeds, regardless of the feature engineering conducted over the input variable. The successive improvement of the feature engineering performed has already been shown in **(2.2)**; the same principle follows in the case of SVMs. Therefore, this will not be shown.

Figure 3(a), 3(b) present the performance of the various approaches. As stated **(2.2)**, a linear basis provides a very reasonable fit. Therefore, as verified, the performance of the Polynomial Kernel is surpassed by the one of the Linear Kernel. The usage of the RBF Kernel, due to its flexibility, induced the best results. We can therefore achieve a **Test Cross Validation Error of 0.0707 and 0.0562** for the binary and multiclass problems, respectively.



(a) Multiclass SVM.



(b) Binary Class SVM.

Figure 3:

2.4 Neural Networks

Lastly, we tackled feed-forward neural networks. We made use of Matlab's Deep Learning Tool Box to implement this particular model.

In the case of this model, the difference between the multiclass classification problem and the binary classification problem resumes to the number of neurons in the output layer, each is respectively four and two.

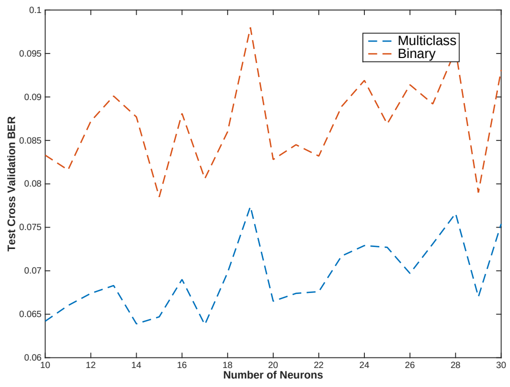
Regarding the neural network architecture, we explored several configurations in terms of number of nodes and number of hidden layers, having the sigmoid function as activation function and the softmax function as link function, since we are dealing with a multiclass classification problem.

We reaching the conclusion that the neural network architecture that performed the best consisted of a single hidden-layer neural network of 14 neurons. Even though the increasing number of hidden layers enables to express the output as any function of the input, it also conveys the risk of the model overfit. Hence, a single hidden-layer model can reasonably well explain the input output relationship. Regarding the number of neurons of this layer, it is known that a low number of neurons in the hidden layers implies loss of important information, while a higher number of neurons can prevent convergence. This balance is illustrated in Figure 4(a). The convergence of the algorithm is function of the number of epochs performed. We ran our models under 15 epochs, which conveys convergence while preventing to high computational cost. The chosen learning rate was 0.1.

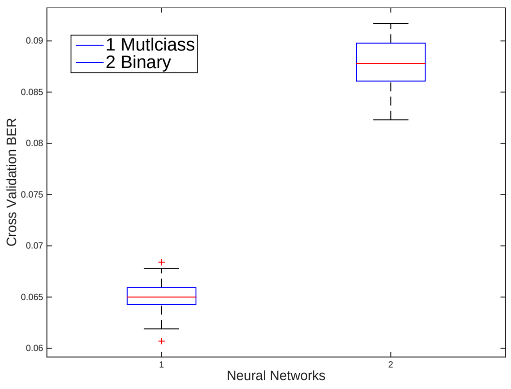
The models were trained using 4-Fold Cross Validation and ran over 25 randomly picked seeds. The performance of the best case is presented in Figure 4(b). We can therefore achieve a **Test Cross Validation Error of 0.0878 and 0.0649** for the binary and multiclass problems, respectively.

3 Summary and Conclusion

Our best model for both the multiclass and binary problems results of the application of multi-class SVMs using RBF as Kernel, after performing Feature Extraction (1.2), (1.3) and (1.4) over *CNN* Features alone. Performing PCA factorization over the training set allowed the use of *CNN* features and to severely decrease computational time; removing outlier through K-Means Algorithm implied that our training algorithms were not influenced by not representative class members; tackling the fact that our dataset was imbalanced, through Random Over Sampling, meant our algorithms were not biased towards the majority classes. The best values for hyper parameters were found through grid search, traducing a balance between the number of support vector and their sphere of influence and consequently optimizing the bias-variance tradeoff. Figure 5(a) represents the test error distribution over the whole training set for our best model. Then, the



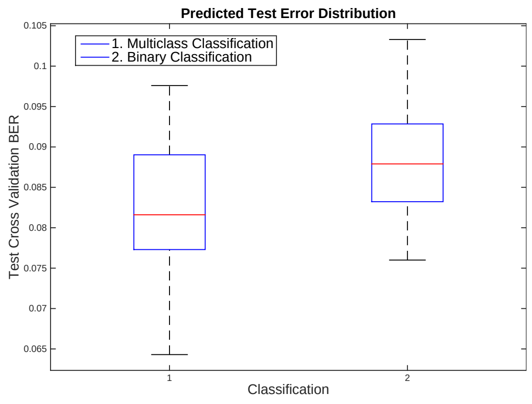
(a) Neural Networks.



(b) Neural Networks.

Figure 4:

predicted Test Error for Multiclass is **0.0825 (with standart deviation of 0.0077)**; for Binary, it is **0.0875 (with standart deviation of 0.069)**.



(a) Predicted Test Error Distribution.

Figure 5:

4 Implementation Details

4.1 PCA factorization

For the case of *HOG* features, it is possible to apply Matlab's own SVD function to extract the singular value decomposition. However, for the case of *CNN* features, its dimensionality is greater than what Matlab's own SVD function can support. Therefore, we recurred to the function LMSVD [5], which computes a truncated dominant singular value decomposition. The function was initialized so that it ran over the maximum number of iterations (300) and with minimum tolerance (1×10^{-8}).

4.2 Implementation of K-Means

A self-implementation of K-Means was design. Regarding the initial conditions, for our propose of use, we computed the initial center to be such that:

$\mu_{\mathbf{k}}{}^{\mathbf{initial}} = \frac{\mathbf{1}}{\mathbf{N_k}} \sum_{\mathbf{x_n} \in \mathbf{C_k}} \mathbf{x_n}$, where N_k is the number of points belongs to Class k.

[5]See Reference

Having done this, the algorithm converges in less than 10 iterations. Still, 10 has taken as default value for the number of iterations

4.3 ADASYN for Algorithm Sampling

ADASYN algorithm's goal is to improve class balance by creating new examples from the minority class via linear interpolation between minority class examples. It is an extension of SMOTE method (Synthetic Minority Oversampling Technique). [6]

4.4 Stochastic Gradient Descent and Learning Rate

Stochastic Gradient Descent (SGD) consists of the following algorithm:

Algorithm 1 Stochastic Gradient Descent (SGD)

```
1: procedure SGD( )
2:     Repeat until convergence
3:     Randomly shuffle the dataset
4:     for <each minibatch> do
5:         <$\boldsymbol{\beta} = \boldsymbol{\beta} - \alpha \cdot \bigtriangledown_{1,...,K}\mathcal{E}(\boldsymbol{\beta})$>
6:     end for
7: end procedure
```

While Batch Gradient Descent (BGD) computes the gradient using the whole dataset, SGD computes the gradient as the sum local gradients over randomly picked minibatches over the all data set. Consequently, as cost functions as generally not smooth, BGD is much more sensible to local minimum, when compared to SGD, which avoids poor local minimum and advance towards a hopefully more optimal region. Furthermore, is computational cost is much lower, even though more iterations are need to achieve convergence. The minibatch size was set to 20 data points. The learning rate α was chosen to be strictly decrescent towards 0, so that the algorithm increases sensible as it approaches the global minimum. To ensure convergence, α was adapted to each model in particular.

Appendix A

Recall the posterior probabilities are given by the softmax transform of linear combinations of the input features:

$$P(y_n = C_k|\mathbf{x_n}) = \sigma(\eta_{nk})_k = \frac{exp(\eta_{nk})}{\sum_j(exp(\eta_{nj}))}$$

where $\eta_{nk} = \tilde{\mathbf{x}}_n^T\boldsymbol{\beta}_k$ and $\sigma(\eta_{nk})_k$ is the softmax function.

The error function is given by the negative logarithm of the likelihood function:

$$\mathcal{E}(\boldsymbol{\beta}) \quad = \quad -log\, p(\mathbf{Y}|\boldsymbol{\beta_1}, ..., \boldsymbol{\beta_K}) \quad \overset{\textbf{(Independence)}}{=} \quad \mathbf{-log \prod_{n=1}^{N} \prod_{k=1}^{K} [P(C_k|\boldsymbol{\beta_k})]^{r_{nk}}}$$
$$= -\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \cdot log\, P(C_k|\boldsymbol{\beta}_k) = -\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \cdot log\,\sigma(\eta_{nk})_k$$

The gradient in respect to each of the parameters beta(k) is given by:

$$\bigtriangledown_j\mathcal{E}(\boldsymbol{\beta}) = \frac{\partial}{\partial\boldsymbol{\beta}_j}(-\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \cdot log\,\sigma(\eta_{nk})_k) = -\sum_{n=1}^{N} r_{nj} \cdot \frac{\partial(log\,\sigma(\eta_{nj})_j)}{\partial\boldsymbol{\beta}_j}$$
$$= -\sum_{n=1}^{N} r_{nj} \cdot \frac{1}{\sigma(\eta_{nj})_j} \cdot \frac{\partial\sigma(\eta_{nj})_j}{\boldsymbol{\beta}_j} = -\sum_{n=1}^{N} r_{nj} \cdot \frac{1}{\sigma(\eta_{nj})_j} \cdot \frac{\partial\sigma(\eta_{nj})_j}{\partial\eta_{nj}} \cdot \frac{\partial\eta_{nj}}{\boldsymbol{\beta}_j}$$
$$= \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \cdot (\sigma(\eta_{nj})_j - y_{nj})$$

We can then find the optimal parameters $\boldsymbol{\beta}$ by making use of gradient descent:

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha \cdot \bigtriangledown_{1,...,K}\mathcal{E}(\boldsymbol{\beta}^{(k)})$$

[6] See Reference

Reference

[1] Chawla, Sanjay, and Aristides Gionis. ”k-means-: A Unified Approach to Clustering and Outlier Detection.” SDM. 2013.

[2] Liu, Wei, Gang Hua, and John R. Smith. ”Unsupervised one-class learning for automatic outlier removal.” Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. IEEE, 2014.

[3] Class Imbalance Problem: http://www.chioka.in/class-imbalance-problem/

[4] Xin Liu, Zaiwen Wen and Yin Zhang, Limited Memory Block Krylov Subspace Optimization for Computing Dominant Singular Value Decompositions, SIAM Journal on Scientific Computing, 35-3 (2013), A1641-A1668.

[5] Caputo B, Sim K, Furesjo F, Smola A (2002). Appearance-based Object Recognition using SVMs: Which Kernel Should I Use

[6] Convolutional Neural Network : http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[7] Neural Networks: http://neuralnetworksanddeeplearning.com/chap5.html

[8] He, Haibo, et al. ”ADASYN: Adaptive synthetic sampling approach for imbalanced learning.” Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on. IEEE, 2008.

[9] General Overview: ”Pattern Recognition and Machine Learning” by Bishop