# Project-II by Group MexicoCity

**Kevin Serrano**
EPFL
kevin.serrano@epfl.ch

**Youssef El Baba**
EPFL
youssef.baba@epfl.ch

**Alexandre Helfre**
EPFL
alexandre.helfre@epfl.ch

## Abstract

## 1 Exploratory Data analysis

For the person detection dataset, there are 8545 images and a feature vector of dimensionality 9360 for each. The images are 3-channel (RGB) images, and the features are simply a vectorized form the image HOG features. Clearly this is a fat data matrix, even more so if we are going to split it into train/test datasets. Anyway, considering the huge number D, we are tempted to do some sort dimensionality reduction if we are to avoid extremely complex models and overfitting. This hints to the possibility of using PCA for this dataset. Or alternatively we can use neural networks, with the help of the Deep Learning Toolbox (kindly indicated by the course staff). We can also try using simple tools like $k-$means with $K = 2$ or SVM.

On initial visual inspection of the images and their corresponding features, it does seem that the HOG features for images with people are of a somewhat different nature. The various blocks containing the $(W \cdot W)$ feature blocks seem to have a much more varied orientation than for the images without people.

As to the music recommendation dataset, there are a total of 1774 users and 15082 artists. Each pair of (user, artist) is a data element, and the corresponding listening count number is the data.. If we change our convention to pairs of (user, user) we get another feature vector of size D=1 that is the friendship relation (currently given as a binary number, indicating a friendship existance). The format of the data set is peculiar with respect to usual matrices containing N rows of D columns (features) each. We can see that some users listen practically solely for a single artist with a high count when inspecting the provided $Y_{\text{train}}$ matrix.

On a different note, it seems daunting at first to predict specific count numbers for each (user,artist) pair, as usually the recommendation in such systems are the k most likeable artists for a specific user. One way to go is to predict, using a computed probability for each (user, artist), a sort of score that is, a count that will be simply the maximum number of counts of the user when the score is 1, and adjusted linearly with the score as the coefficient.

## 2 People Detection

### 2.1 Neural Networks

When playing with the person detection code given to us, using the deep learning toolbox, the training error obtained was 0.745, which is approx. 75

Continuing with neural networks testing, we tried adding a dropout fraction of 0.3 and see if it could improve the error rates. The stable (after 25 epochs) mean-square error (obtained using the native plot of the Deep Learning Toolbox nntrain function) for the methods without and then with dropout was 0.005774 and 0.006937 respectively. Oddly enough though, average TPRs for these methods are 0.75 and 0.787 respectively. A second run with another seed for the Matlab random number generator gave other final MSEs and TPRs, this is because NN are very sensitive to the initial seed. So the seed for all further trials was manually set to 8339 (given by the course staff) . Adding a dropout seems to enhance our TPR,, but gains are somewhat mitigated for bigger FPRs. The batchsize for these methods (a parameter in the Deep Learning toolbox) was also tampered with, when we tried to take the whole training/data as a batch (batch size 4272), however this turned out to be a bad idea as the convergence of the MSE cost (on the toolboxs training graphs) only attained 0.014 at the lowest level, much higher than with a batchsize of 100. Then we tried changing the internal activation functions from tanh to sigmoids, with an improvement of the MSE / TPR to 0.002975 and 0.795, which was quite remarkable. The last 2 variants to be tried were simple NN with $L_2$ weight penalty equal to 1/1000 and simple NN with both sigmoid activations and a dropout of 0.3, which respectively gave MSE / TPR values 0.006758 / 0.772 of 0.00404 / 0.844. These results

In addition to that, logistic regression was given a try, and it gave a TPR of 0.7521. Then Penalized logistic regression with a lambda of 1/50 also gave the same TPR. The $\lambda$ for the logistic regression clearly wasnt adequate.

One clear conclusion that can be drawn here is that internal sigmoid activation functions, instead of tanh, help a lot. They are going to be used for the final model then (if using NN).

Clearly, each of these additions to the basic NN model improves things, on its own, a bit. However when they are applied together, things improve considerably (as indicated by the last cited and quite high TPRs). Of course, the values for these parameters (dropout and $L_2$ penalty) were chosen arbitrarily. Similarly, Penalized Logistic regression can have a different optimal lambda, which should be cross-validated, and might result in higher TPR.

### 2.2 Penalize Logistic regression (cross-validation for $\lambda$

Penalized logistic regression was cross-validated (for the lambda parameter) to see which setting gives the best estimated TPR. The error metric taken was 1-TPR, which made simple sense. $\lambda$ values between $10^{-3}$ and $10^2$ were tested, 15 of them, and using 15 different seeds. So a total of $15 \cdot 15 = 225$ gradient descent runs for PLR were tried, giving the following plot (Figure 1) It seems that, roughly, the more we increase lambda, the better our test error equivalent (1-TPR, which is actually the False Negative Rate) is. For $\lambda$ = 44, we deem it the test error no longer decreases

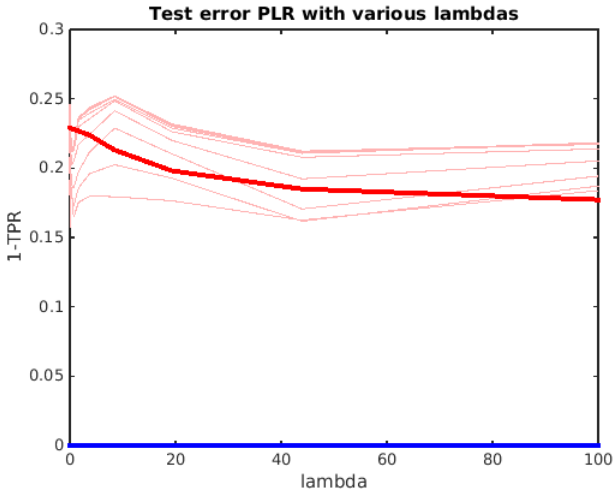



Figure 1: Test error PLR with various $\lambda$

significantly and its sufficiently good at 0.1849 (this corresponds to a TPR of 0.8151). This would be the best possible variant in the Logistic regression family to use. It remains to be seen at this point, however, if other ML methods can do better, especially after all their variants have been cross-validated (NN with various dropouts and $L_2$ weight penalties for instance). (red curves show the 1-TPR for test set, using model trained on a different training set, for each lambda, each curve being the result of a different seed, the bold curve being the average).

### 2.3 $K-$means

We then tried using $k-$means on the same train and test sets used for the previous methods, with identical normalization (essentially how the provided demo script did the split). The MATLAB $K-$means is particular in the sense that it does not take the actual (true) labels (person / no person) of each point (image) in account when computing its assignment, it simply renames the classes as $1, 2, 3, \cdots, K$. What it does is it simply clusters in a best effort manner the points, and assigns random labels to them. We tried at first to simply compute the MATLAB labels on the test data and reassign them to $-1$ and 1, which gave a TPR of 0.0057 (or 0.57 %), very low for any practical purpose to say the least. We then tried to re-wrap the $k-$means function in another one generating a "model" (centroids + labels of the centroids computed as the mean of the labels of their assigned points) which now uses the true labels of the points. And another function which, as is typical of other methods, predicts the assignments of new points (using euclidean distance to centroids). This now resembled classical model based methods, however it also failed miserably with a TPR of $5.8 * 10^{-4}$. This can be explained by the fact that the cluster of no-person images is much, much larger than the one with person-containing images, hence when assigning the labels to the cluster means they consistently get the label "no-person" making the result a static label $-1$. As taking the mean label of the cluster points and assigning it to the cluster mean is an integral part of the $K-$means algorithm, this ML method is probably not the one to use here.

### 2.4 Support Vector Machine

The next method we tried to apply on the people detection dataset was SVM. We used the `fitcsvm` and `predict` functions of MATLAB, to be able to obtain posteriors probabilities and not just hard assignments. We tested plain old SVM, with a linear kernel, and by taking a slight transform of the posterior we were able to obtain a TPR of 80.59 %. We tried varying the $C$ constant parameter, trying values of 0.1, 0.5, 2 and 5, however they all gave the same TPR. We then tried using an RBF kernel, with $\sigma$ values of 0.5, 1 and 2, but they all gave a very low, consistent TPR of 0.0582 %. This is very similar to what we get by simply predicting a $-1$ (no-person) for all the images. The prospects for the basic variant of this method provided good performance, but its lack of flexibility for further improvement makes us believe we should opt for another ML method for this dataset instead.

## 3 Music recommendation

### 3.1 Histogram

Following is a the histogram of the counts for the Music recommendation dataset (Figure 3(a)). The max count is 352698, which is huge. Moreover the number of counts $> 5000$ is rather small. Thats why were only showing the bins for the counts that are smaller or equal to 5000, the rest being generally present once or twice at most.

After observing the histogram of the counts, as proposed by the course staff, we can see that despite the high maximal count, most of the data is concentrated in the low-count range (1-3500). Going out on a limb here we might consider any count quite outside of this interval to be an outlier, however things are a bit nuanced: if we start removing outliers from the data (which is already sparse) we run into the problem of deciding how far the point has to be from this interval to consider it as such. That decision is very difficult to make, especially considering that music listening - as any human-behaviour cannot be expected to always follow mathematical rules. What about poisson distribution for count?

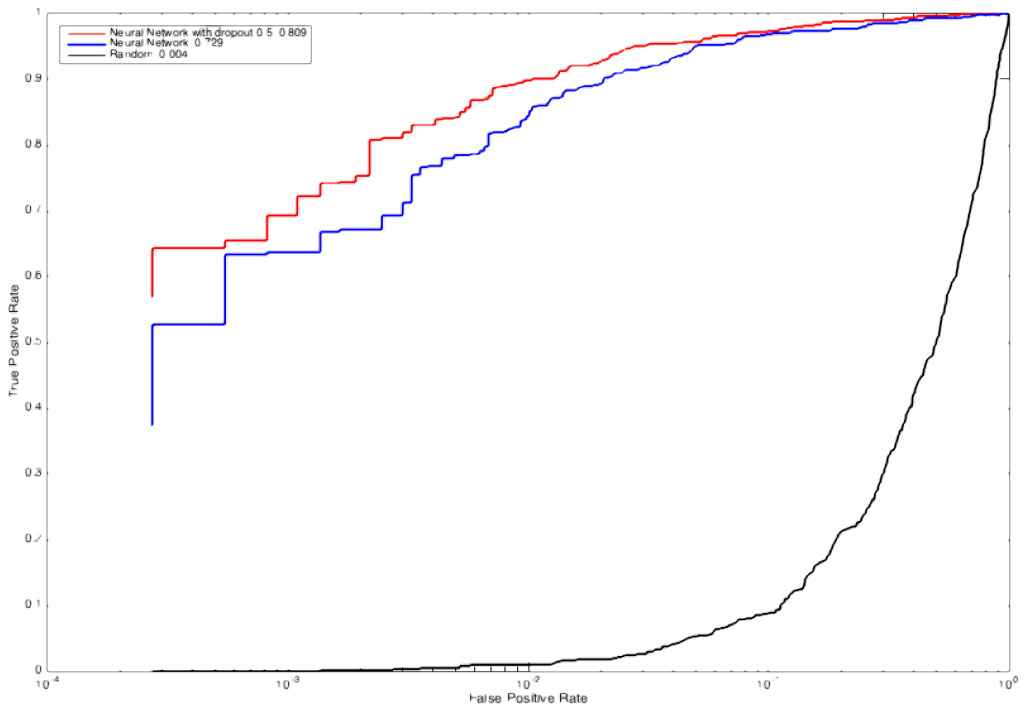
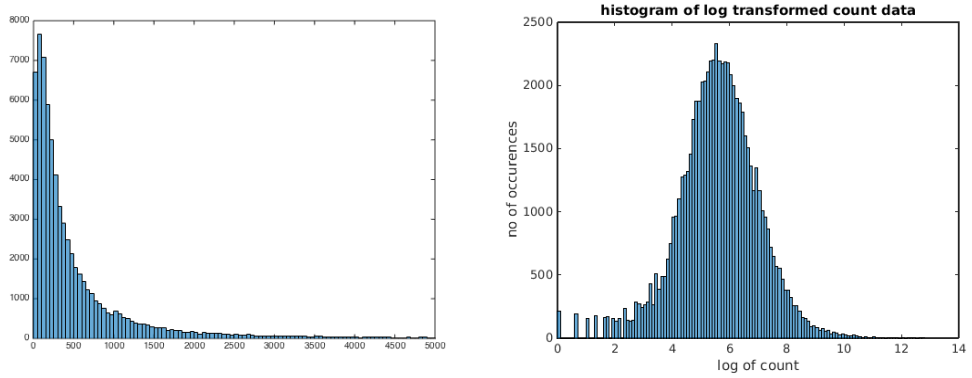



Figure 2: Comparison of different methods for People detection




(a) Histogram of counts

(b) Histogram of log transformed counts

Figure 3: Histograms

A second conjecture is that, inside this interval, most counts are still concentrated at the lowest range. Its as if the histogram of the counts decays exponentially fast, which is why taking the logarithm (of the counts) would change the $x$ axis scaling in some sorts and produce a more bell shaped curve, i.e. a Gaussian. This is experimentally confirmed by taking the histogram of the counts (full counts, not restricted to the interval, see Figure 3(a)) and then taking the histogram of the log of the counts, which does correspond to a Gaussian, as shown in Figure 3(b). This shows potential for such a transform to be used before applying ML methods, especially those one imposing Gaussian-distributed restrictions on their data.

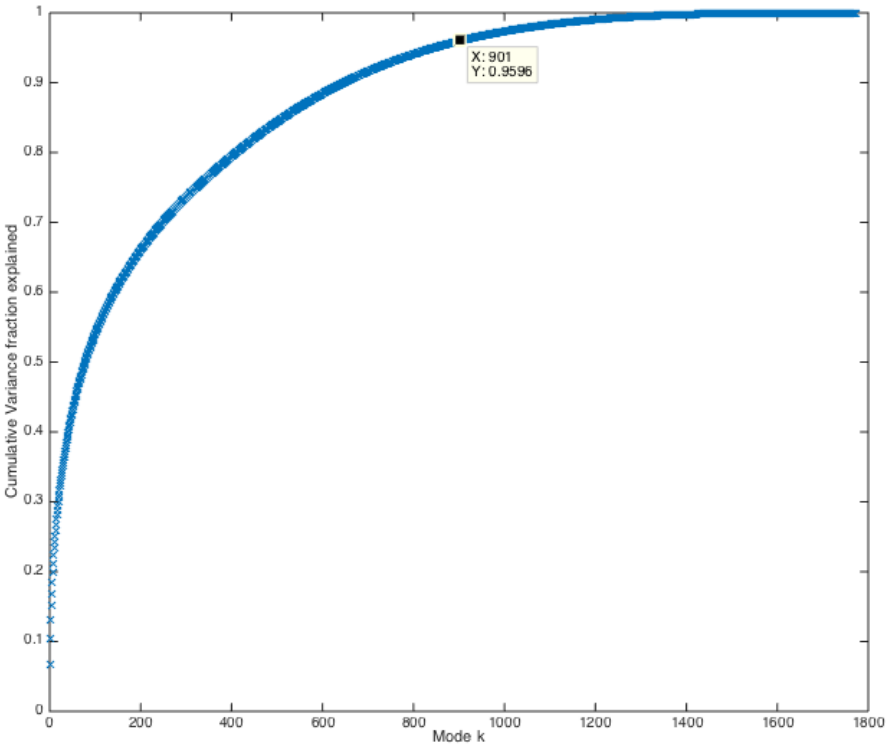



Figure 4: Cumulative variance fraction explained

### 3.2 PCA/SVD

We ran the PCA algorithm on the full-format $Y_{train}$ data. Nevertheless the ALS algo (PCA with the ALS option) does not work with this matrix, as it seems that $Y_{train}$ is a too large a matrix even if it is sparse in practice.

Therefore we applied the SVD decomposition on $Y_{train}$ to see if we can reduce in practice reduce the dimensionality of our problem. To illustrate that possibility, following is the plot (Figure 4) of the cumulative fraction of variance explained vs the modes. (For mode 1 we use singular value $s_1$ and its corresponding vector, for mode 2 we use $s_1$ and $s_2$ and their etc, increasing the rank of our approximation). The result can be summarized in that the 900 most significant vectors (highest singular value) explains 95% of the datas variance, which means they can be roughly sufficient to use as a basis for training ML models and predictions.

### 3.3 Basic mean-based predictions

One of the simplest prediction method is to use averages to predict. The first thing we have done is to compute the average listening per artist on our training set. Then using only the positive values in our test set, we can compute the RMSE between our prediction and the real values. If we dont apply any transformation to the data, we obtain a RMSE of 6400. We can see that this is quite a high number but nevertheless expected since the RMSE supposes an Gaussian input and that the average method is not a good one.

Then we applied the log transformation to obtain a more Gaussian input as seen in Figure 3(b). We then compute the RMSE based on the same principle as before and rescale the result with the exponential to obtain an RMSE of 4130, which is already much better.

### 3.4 Collaborative-filtering

## 4 Summary

### Acknowledgments

### References

Kevin Murphy, Machine Learning : A Probabilistic Perspective *The MIT Press Cambridge, Massachusetts London, England,2012*
Trevor Hastie,Robert Tibshirani,Jerome Friedman : The Elements of Statistical Learning. Data Mining,Inference,and Prediction. *Springer, 2008*
Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani:An Introduction to Statistical Learning.*Springer, 2013*