Project-II by Group Rome

Cimen Gokcen
EPFL
gokcen.cimen@epfl.ch

Angelopoulos Vasileios
EPFL
vasileios.angelopoulos@epfl.ch

Petrescu Viviana
EPFL
viviana.petrescu@epfl.ch

Abstract

This report presents experimental results and performance comparison of common machine learning methods on two different tasks. The first task is a computer vision problem, a people detection task. We implemented classifiers that are able to predict to a certain degree if an image contains a standing person or not. The performance was measured using ROC curves. The second task is a music recommendation challenge. For this problem, we need to predict how many times an old or a completely new user will listen to a certain artist, based on the available listening history of the users and their friendship information. The performance of the various methods was measured using the mean average error of the transformed data.

1 Music Recommendation

1.1 Music recommendation

Collaborative filtering is the part of recommender systems that predicts users' preferences for particular items. The major challenge in predicting users'listening counts, as in our task, is that the available user-artist entries are usually too few and sometimes a method's performance relies on a good initial estimation of the unknown entries. We therefor decided to implement besides the common KNN, Kmeans also ALS[cite here], which uses only the known listening counts and avoids dependency on initial estimations.

1.2 Data description

The training data consists in a matrix $Ytrain$ of size 1774x15082, corresponding to 1774 users and 15802 artists. Entry $Ytrain(i, j)$ expresses how many times user i has listened to artist j. An entry of 0 means we do not have information for that $(user, artist, count)$ triple. We are also given the friendship graph of the 1774 users stored as an adjacency matrix.

1.3 Exploratory Data Analysis

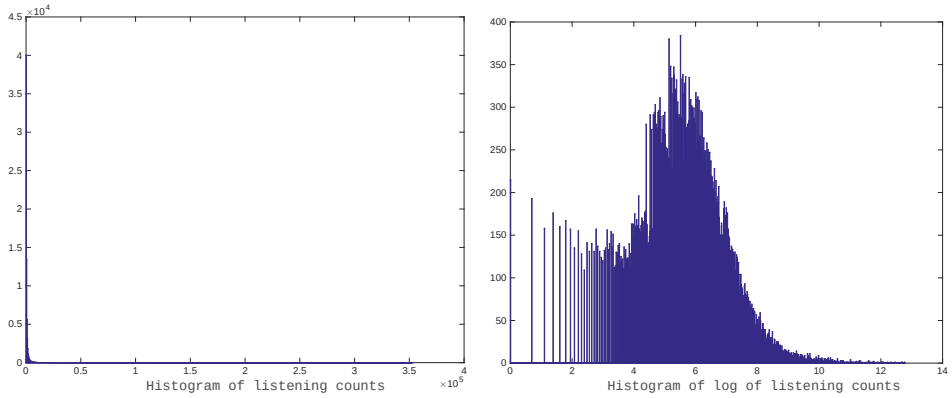
The matrix $Ytrain$ is very sparse with a density of only 0.0026%, corresponding to $69617(user, artist, count)$ triples. The variance of the entries' values is very high, the maximum being 352698 and average listening count per user and per artist of 5.52 and 5.46 respectively. There were also 1262 artists for which no information was provided.

A histogram of all the listening counts, shown in Fig1 $a)$ tells us that the known entries follow a heavy tail distribution. The long tail contains a small number of popular items, the well-known artists, and the rest are located in the heavy tail. One method to transform skewed data such that it becomes more gaussian distributed is to use the Box-Cox transform

$$
\begin{aligned}
data(\lambda) \quad &= log(data), \lambda = 0 \\
&= \frac{data^{\lambda}-1}{\lambda}, \lambda \neq 0
\end{aligned}
$$

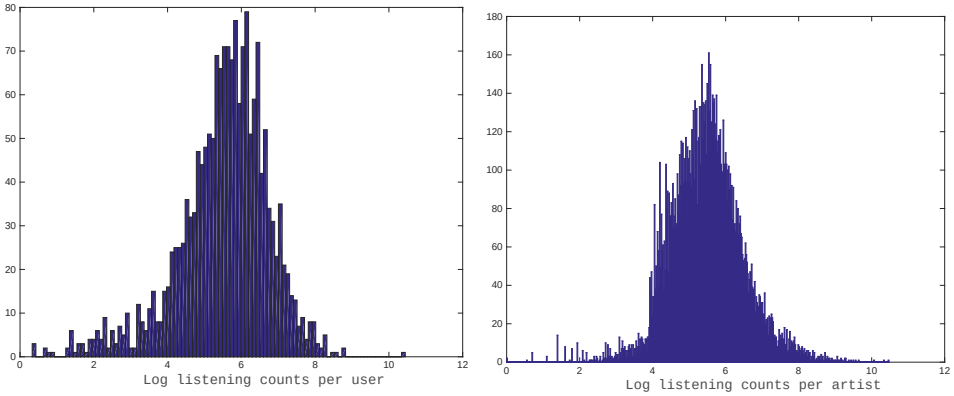
In our case, we can choose $\lambda = 0$ because our values are very high and positive. This transformation will make the distances between listening counts much smaller and will reduce the influence on error of the (user,artist,count) triples in the long tail.

The results after data transformation can be seen in Fig 2. The distribution of the user counts and of the artist counts are closer to a normal distribution. We notice there are some far away entries with values greater than 14. These will have a higher impact on the untransformed data. An off by 1 error here will have more impact than on the small entries.



(a) Before data transformation (b) After data transformation

Figure 1: Distribution of all listening counts



(a) (b)

Figure 2: Distribution per user (a) and per artist (b) log of listening count distribution

1.4 Task 1

In all our experiments we used 10-fold cross validation and repeated the experiments twice. For the first task, we randomly omit 10 entries for every artist. Splitting the data was more difficult since we needed to make sure we do not remove all the entries for an artist. If an artist has m entries with $m < 10$, then we keep $m - 1$ entries for testing, to still have one element for training.

1.5 Baseline

There are three simple basic predictions one can try: the global average count, the average count per user and the average count per artist prediction. All of these methods give similar MAE results: 1.1117(pm 0.0079), 0.64(pm 0.0042) and 1.2134(pm 0.0092). We note that the final MAE is composed of various values, small and large. In Fig we can see a ploto of the MAE error terms in log format.

1.5.1 KNN

K-nearest neighbors (KNN) algorithm is the first approach. In this approach, compared to other methods, there is no model and model parameters. We determine the listening count of a user on a artist by estimating the similarities of users, and considering k listening counts of k similar users on the artist. We use pearson metric between two users:

$$similarity(u1, u2) = \frac{\sum_{i \in u1 \bigcap u2}(R_{u1,i} - \bar{R_u}1)(R_{u2,i} - \bar{R_u}2)}{\sqrt{\sum_{i \in u1 \bigcap us2}(R_{u1,i} - \bar{R_{u1}})^2 \sum_{i \in u1 \bigcap u2}(R_{u2,i} - \bar{R_{u2}})^2}} \quad (1)$$

In recommendation phase, the following formula gives the predicted listening count of a user for an artist:

$$p(u, i) = \frac{\sum_{k \in N} similarity(u, k)(R_{k,i} - \bar{R_k})}{\sum_{k \in N} |similarity(u, k)|} \quad (2)$$

The number of nearest neighbor, k, in KNN method is the only parameter that should be set. The best value for k depends upon the data. Therefore, we selected the good k value by cross-validation technique. Using 10 among $[5, 10, 15, 20, 30]$ gives the best results for KNN as can be seen from table 1.5.1.

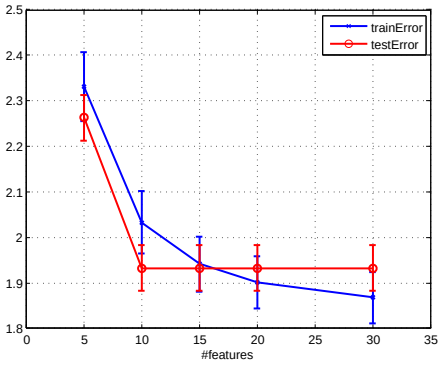


Figure 3: Comparison of Train and Test errors with different k values

k	MAE train	MAE test
5	2.33 ($\pm$ 0.05)	2.26 ($\pm$ 0.076)
10	2.03 ($\pm$ 0.049)	1.93 ($\pm$ 0.068)
15	1.94 ($\pm$ 0.049)	1.93 ($\pm$ 0.060)
20	1.90 ($\pm$ 0.049)	1.93 ($\pm$ 0.057)
30	1.86 ($\pm$ 0.049)	1.93($\pm$ 0.056)

Table 1: Estimated Train and Test MAE for K-Nearest Neighbors

Increasing the k value up to 10 reduces both test and train error. After 10, increasing the value of k doesn't change the test error but continues to reduce the train error for our data. Increasing the k value cause overfitting where test error is very big than train error, so we selected 15 as an optimal value for k. The results gave a MAE 1.9 for the test error.

1.5.2 ALS

We do not review here the details of ALS algorithm, the reader can consult the paper[cite] beforehand, since this was not the goal of this report. TODO

Using value of 20 for features and $lambdain[0.01, 0.05, 0.1, 0.5, 1]$ we obtain the results in table 1.5.2 using 10-fold cross validation. The experiments were repeated twice with different seed. We note that we stopped the update steps in the algorithm only after 5 iterations because the update step was very computational intensive.

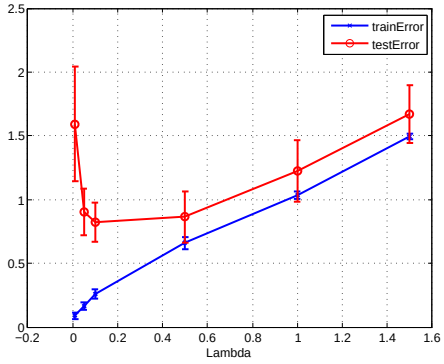


Figure 4: Comparison of Train and Test errors with different lambda values

lamb	MAE train	MAE test
0.01	0.088 (± 0.0008)	1.591 (± 0.015)
0.05	0.163 (± 895)	0.901 (± 0.0061)
0.1	0.259 (± 851)	0.822 (± 0.0012)
0.5	0.660 (± 851)	0.864 (± 0.0016)
1	1.035 (± 842)	1.222(± 0.0010)
1.5	1.493 (± 837)	1.670(± 0.0007)

Table 2: Estimated Train and Test MAE for different lambdas

We can see that a value of 0.05 for λ is a reasonable choice, so we selected lambda to be 0.05 for the next experiments.

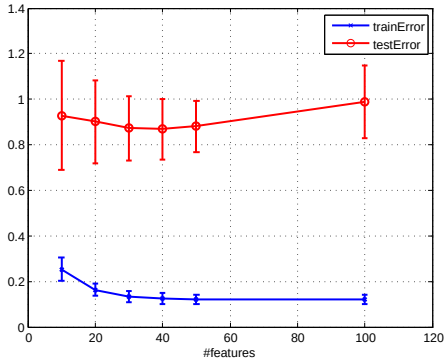


Figure 5: Comparison of Train and Test errors with different features values

feat	MAE train	MAE test
10	0.253 (± 0.0017)	0.927 (± 0.0080)
20	0.163 (± 0.0009)	0.901 (± 0.0061)
30	0.133 (± 0.0009)	0.873 (± 0.0047)
40	0.124 (± 0.0008)	0.868 (± 0.0044)
50	0.121 (± 0.0007)	0.880(± 0.0038)
100	0.122 (± 0.0007)	0.989(± 0.0053)

Table 3: Estimated Train and Test MAE for different features

We varied the number of features from $10, 20, 30, 40, 50, 100$ with $\lambda = 0.05$ and repeated the experiments with different seed. Giving train error from 0.122 increasing up to 0.253 while the test MAE is 0.989 for maximum and 0.88 for minimum. Even though value 50 for features is best for train error, we choose 40 because it give better results for both test and train error. The results gave a MAE less than 0.8 for the test error when λ is 0.05, and feature count is 40.

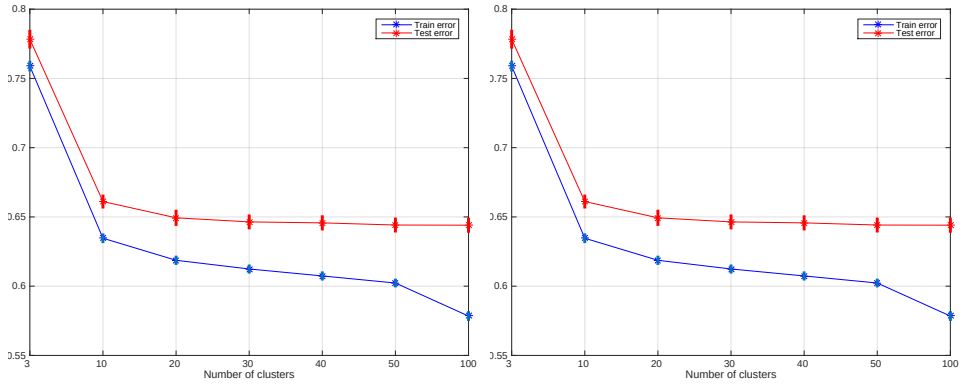
1.5.3 Kmeans

Although the number of artists is 10 times larger than the number of users, we chose to implement Kmeans with clustering of users instead of artists so that we can use the clusters obtained here also in Task 2.

The missing entries in the matrix were initialized with the user average, although that did not have a huge impact on our results. We then sorted the users according to their average score and initialized the clusters with equally spaced samples from the sorted users array. The goal of this cluster initialization was to have a more equilibrated assignment of users to clusters and to make sure we have clusters from both highly active users and also from the less active.

Inspired by the friendship graph information, we tried Kmeans with varying values from [3,10,20,30,40,50,100]. We plot the mean train and test error using 10 fold cross validation in Fig 6. Using only 3 clusters both the train and the test error was high, while for 100 clusters the difference between train and test error became larger, giving signs of overfitting.

We noticed also the fact that sometimes the training error of Kmeans (reported by taking exp of data and RMSE) had small fluctuations and it was not always decreasig as the algorithm convergence properties would expect. This is due to the fact that Kmeans miminizes squared error but our cost is a little different since we transform the data using exp.



(a) (b)

Figure 6: bla bla

In all the experiments with more than 20 clusters obtained a test MAE close to 0.64,. This was our best test MAE so far. For computational reasons, we picked 20 clusters for the next task.

We also tried KMeans where in the update step, only the entries for artists that were known were updated. We expected this to perform better, but the algorithm had problems with overfitting, always giving a train error less than 0.4 (much less than the normal Kmeans) but a test error a bit over 0.7 (higher than the normal KMeans). This happened for all the cluster sizes we experimented with.

1.6 Task 2

In this task we make predictions for a set of new users using their friendship information with a set of old users for which partial listening history is available. In other words, we try to see if there is a correlation between users' friendship and users' preference in music.

1.6.1 Friendship information

The initial friendship graph $Gtrain$ of 1776 nodes and 22904 edges contained 22 connected components, but the majority of the components contained just a few nodes. Using the Gephi [1] tool we were able to find some properties of the graph such as the number of communities, 32. Out of these 32 communities, only 8 communities had more than 150 members. These statistics were run so that we get an idea of the number of friendship clusters present. The number of connected components and of communities are similar with our choice of 20-30 clusters in KMeans .

We have tried two approaches using the friendship information, one based on the mean average per user and one based on kmeans clustering of users.

1.6.2 Baseline Methods

As a reference method, we predict for a missing value the global average of all the available counts, which gave us a MAE of 1.078($\pm$0,044). Taking the average per artist gave us a MAE of 1.767($\pm$ 0.09), significantly worse than the above.

[1] http://gephi.github.io/

1.7 Average Friend Method

We sum up this approach as "You are the average of your friends". In mathematical terms, a prediction for a new user $Ystrong(u, a)$ is computed as the average count of its friends'listening counts $Ytrain(f, a)$ for that artist or global average count for an user without friends (sad).

$$\begin{aligned} Ystrong(u,a) &= \frac{\sum_{f \in Friends(u), Ytrain(f,a) \neq 0} Ytrain(f,a)}{n\_fa}, n\_fa \neq 0 \\ &= global\_average, n\_fa = 0 \end{aligned}$$

where n_fa is the cardinality of the set $\{Ytrain(f, a) \neq 0, f \in Friends(u)\}$

For this setup we obtain 1.52 (pm0.291) which is smaller than our global average baseline prediction. If we also take into account the friends of the friends of user u we obtain a MAE of 1.08($\pm$ 0.041). This result is comparable with the global average result.

1.7.1 KMeans

We cluster the users into 20 groups using the KMeans setup from Task 1 and repeat the experiments. For a new user, we predict the count as the mean of its friends' cluster. This gave us a MAE of 1.08($\pm$ 0.04). This is similar with the best prediction from the above approach and with the baseline global average. Although KMeans with 20 features behaves similar with the baseline global average, we decided to use this on our newly predicted data.

We mention that we had other unsuccessful experiments where the clusters were updated with the mean of the unique values or using only the most common cluster (test MAE = 1.49), KMeans with 100 clusters (test MAE = 1.55). Same results obtained with 30 friends.

1.8 Summary

Skwes the results, it is more important to have correct predictions on the users with hight counts. Looking back, we could have kep track of the max and median value of the test error to asses the accuracy of our results.

2 People Detection

2.1 Data description

The training data $imgs$ contains 8545 color images of size 105x43. From every image a HOG descriptor 26x10x36 was extracted and converted into a vector of total length 9360. All of these descriptors make up the training data on which we will work on. We noticed that there are more images without people (7308 negative samples) then with people (1237 positive samples).

Our task is to train various classifiers, so that we will be able to detect the presence of people in new images. For this purpose we evaluate five classifiers, measuring the accuracy of their estimations using Receiver Operating Characteristics (ROC) curves.

2.2 Data preprocessing

We assume that there are no outliers, since the images are manually annotated. We work on the extracted features so we normalize the 9360 dim vectors to have 0 mean and standard deviation 1. For all the methods presented below we used 5 fold cross validation.

The methods that we compare are: Naive Bayes classifier, Logistic and Penalized Logistic Regresion, Support Vector Machine (SVM), K-NN, and Neural Network classifier. We note that before the experiments we tried using PCA for reducing the dimensionality of the feature descriptors. The reduction in dimensionality was minimal but at a huge computational cost so we decided to drop it. TODO ask vasilis what was the reduction

2.3 Naive Bayes classifier

In this part we tried to fit a Naive Bayes classifier to our data. For this, we made the assumption that the data is normally distributed and the estimates of the prior probabilities can be estimated empirically from the relative frequencies of the classes in training samples.

The results from the K-fold validation were not very satisfying, since the average TPR was very low (0.031). This may be caused by the strong assumptions that we made in order to fit this model. Naive Bayes model assumes that the processed data follows a specific distribution (in our case Gaussian), but also that the value of a particular feature is uncorrelated with the presence or absence of any other feature, given the variable of the class. In our data, this is possibly not the case, so the classifier cannot fit an appropriate model.

2.4 Logistic and Penalized Logistic Regression

Our goal in this part was to train a simple Logistic Regression model and its Penalized version as baseline methods. We chose a learning parameter $\alpha = 10^{-2}$, since all the values close to this performed well enough for our experiments. Regarding the value λ for penalized logistic regression, all of the values that were used performed the same. Both of the methods give the same results, with average TPR equal to 0.754.

By adding a penalization term, we tried to prevent overfitting by imposing penalty on large fluctuations of the model parameters and reduce the influence of the outliers to our model. The data, as we mentioned before, does not have outliers and the fluctuations at the model parameters are avoided by the nature of our data. This is possibly the reason that both models perform similarly.

2.5 Support Vector Machines

For this part of the experiments we used the LIBSVM 3.20[2] toolbox that supports SVM with various kernels. We performed analysis and K-fold validation on linear-kernel SVM, polynomial-kernel SVM with degrees 2 and 3, radial-basis-function SVM and sigmoid-kernel SVM. In all the models we used the scores that were given by the program to compute the TPR and the FPR. A comparison of the various model is given in Fig 7a.

We can observe that the RBF SVM performs significantly better. This fact can be seen not only by looking at the average TPR, equal to 0.893, but also observing the whole range of the FPR, where the classifier gives better results than all of the other SVM models. The second best in performance is polynomial SVM with degree 2, which has very similar results with the RBF case for FPR greater than $210e{-}3$. The rest of the classifiers perform well for FPR greater than $210e{-}2$, with the exception of the linear case, which has quite good results for lower rates, too. Cubic SVM performed worse than Quadratic SVM probably due to overfitting on the training data.

Moreover, in Table 4 we present the average percentage of the successful classifications that were made during the K-fold validation, as they are given by the program.

SVM kernel	Linear	Quadratic	Cubic	RBF	Sigmoid
Average success rate (%)	94.81	97.22	91.45	98.43	84.68

Table 4: Average percentage for the success rate of various kernels of the SVM classifier.

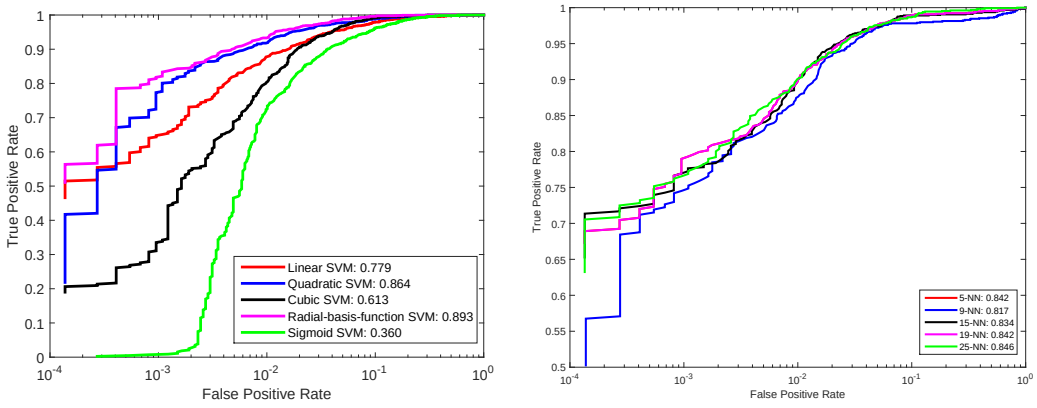
2.6 K-NN classifier

For this experiments we used a KNN library, which we modified a lot in order to compute and return as output not only the class labels, but also the scores of the prediction. We used Euclidean Distance as distance measure between samples and varied the number of neighbours between 5, 9, 15, 19 and 25.

Fig 7b presents the resulting ROC curves that were created using the scores obtained from the K-fold validation runs. There we observe that all of the classifiers show good performance, especially for

[2]http://www.csie.ntu.edu.tw/ cjlin/libsvm/

FPR greater than 10^{-3}. Only the 9-NN classifier seems to perform a bit worse for FPR between 10^{-4} and 10^{-3}. For FPR greater than 10^{-3} the algorithms perform similarly, resulting in almost overlapping curves.



(a) (b)

Figure 7: ROC curves created using: (a) SVM with various kernel types (linear, quadratic, cubic, real-basis-function, sigmoid), (b) K-NN classifier for various number of neighbours for the computations (5, 9, 15, 19, 25).

2.7 Neural Network

Neural Networks are powerfull tools able to learn complex decisions boundaries. Besides the computational cost, their major drawback is the high amount of parameters needed to be tuned for a specific problem, making them harder to train.

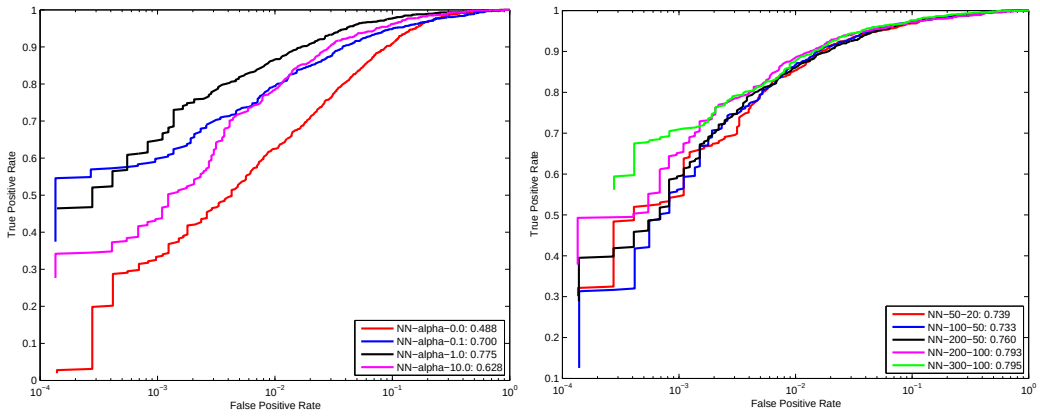
The parameters we considered for our problem were the number of epochs, the learning rate α (parameter in gradient descent update), the number of hidden layers and their size. We fixed the batch size to 50 and the number of epochs to 10, since we experimentally saw that the error starts to converge after 10 passes through the data. The activation functions were also kept fixed to tanh (hyperbolic tangent) and sigmoid at the last layer.

Using cross validation for a network with just one hidden layer of 50 neurons we determined the learning parameter α, which we later used for bigger network setups. Fig.8a shows the ROC curves for α in $[0.01, 0.1, 1, 10]$.

The results for bigger setups can be seen in Fig 8b. Choosing $\alpha = 1$, we experimented with 2 hidden layers of sizes $[50, 20]$, $[100, 50]$, $[200, 50]$, $[200, 100]$, $[300, 50]$. From the results in Fig 8b we could see that the smallest NN performed worse than the others, meaning we can increase the model complexity by adding more neurons per layer or more hidden layers. We could have easily increased and test larger networks if it were not for the time and computational contraints. Not suprisingly, our best setup consists in the biggest network. We see that increasing the size of the network leads to less FPR and better performance in general, meaning we have not yet reached an overfitting point.

2.8 Comparison

To sum up, the models that performed best on the people detection data were SVM (with RBF or Quadratic kernels) followed closely by KNN with 25 neighbours and Euclidean Distance as metric. Although our NN setups performed just a little better than Penalized Logistic Regression, we believe more complex models would have fitted better. We note that our models depend on the discriminative power of the HOG features. Using the same models but with different feature descriptors would have changed the rankings of our best models.



(a) (b)

Figure 8: ROC curves created using: (a) NN with one hidden layer of 50 neurons, α in $[0.01, 0.1, 1, 10]$ (b) NN with two layers of various sizes. In the legend, NN-50-20 corresponds to a network of 50 and respectively 20 neurons for the 1st and 2nd hidden layer.

Acknowledgments

We would like to thank the course teaching assistants that helped us a lot during this project preparation, not only during the exercise session, but also at their office hours. Furthermore, the help that our teacher Mohammad Emtiyaz Khan offered us was very important, and we would want to thank him, too. All of the code used and submitted along with this report was written by equally by all of our group members.