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

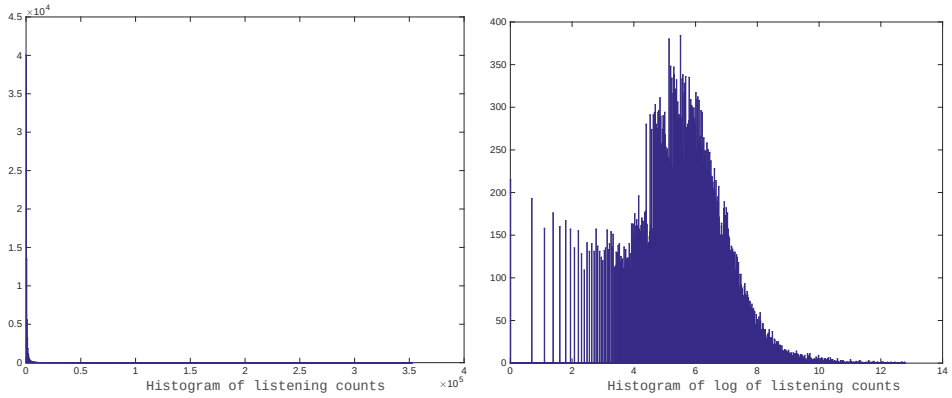
The matrix $Ytrain$ is very sparse with a density of only 0.0026%, corresponding to $69617(user, artist, count)$ triples. The variance of the entries' values is very high, the maximum being 352698 and average listening count per user and per artist of 5.52 and 5.46 respectively. There were also 1262 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

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

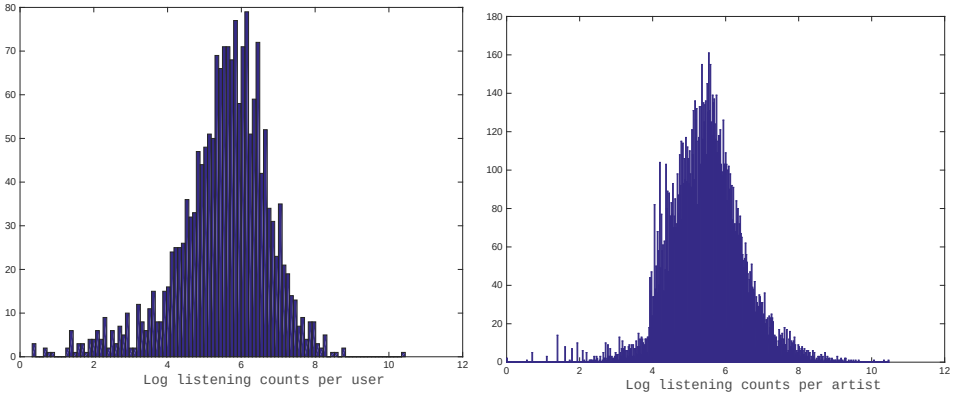
In our case, we can choose $\lambda = 0$ because our values are very high and positive. This transformation will make the distances between listening counts much smaller and will reduce the influence on error of the (user,artist,count) triples in the long tail.

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

TODO

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

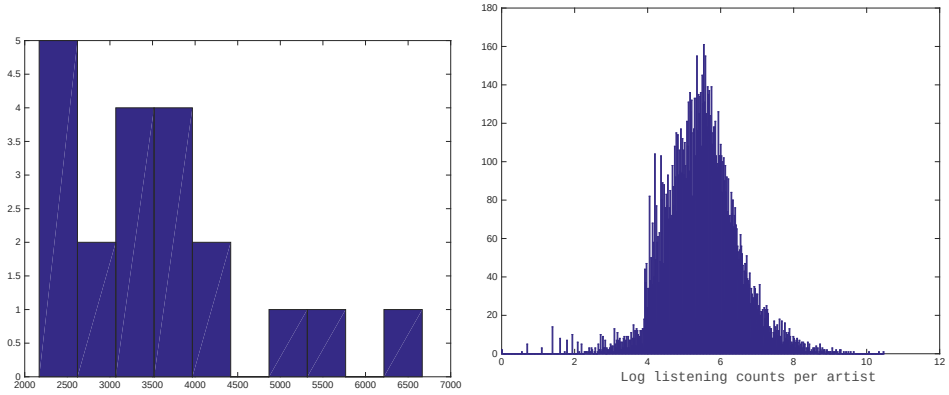
Using 20 features and $lambdain[0.01, 0.05, 0.1, 0.5, 1]$ we obtain the results in table 1.5.2 using 10-fold cross validation .The experiments were repeated twice with different seed. We note that we stopped the update steps in the algorithm only after 5 iterations because the update step was very computational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

lambda	RMSE train	RMSE test
0.01	5075 ($\pm$ 1.0793e+10)	1.13 ($\pm$ 0.05)
0.05	3355 ($\pm$ 895)	1.14 ($\pm$ 0.02)
0.1	3350 ($\pm$ 851)	1.58 ($\pm$ 0.06)
0.5	3341 ($\pm$ 851)	13.93 ($\pm$ 0.86)
1	3387 ($\pm$ 842)	25.30($\pm$ 1.48)
1.5	3416 ($\pm$ 837)	44.25($\pm$ 2.93)

Table 1: Estimated Train and Test RMSE for the two blob models.

We can see that a value of lambda in the middle is a reaasonable choise, so we selected lambda to be 0.1 for the next experiments. Our algorithm did a good job training with a RMSE less than 3 but a bad job on unseen data with RMSE ¿ 3000, meaning it is overfitting the training data and incrasing the value of lambda, the regularization parameter did not help.

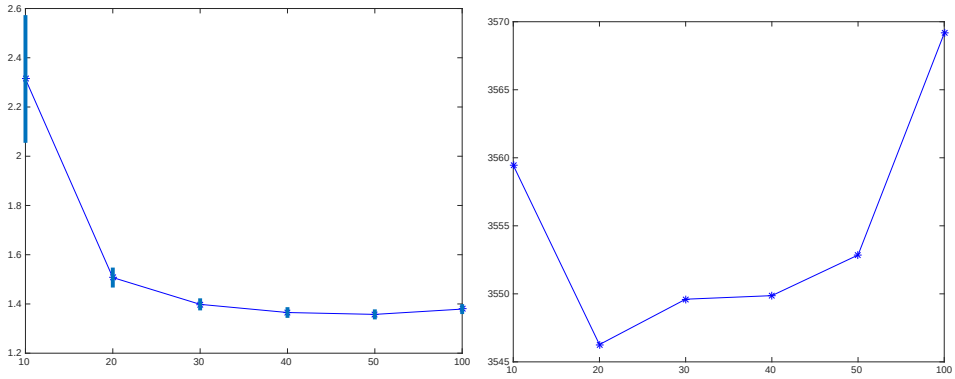
We notice that the test error across the 10 different splits of the cross validation has values from 2000 to up to 6000. This is an artifact of the high listening counts present in the long tail and the randomness of the split.



(a) (b)

Figure 3: Distribution per user (a) and per artist(b) log of listenining count distribution

We varied the number of features from 10,20,30,40,50,100 with $\lambda = 0.1$ and repeteated the experiments with different seed. Giving train between 1.24 increasing up to 2.4 for 19 features. The test RMSE was always in the interval [3569,3546]. None of this results results proved meaningful.



(a) (b)

Figure 4: Train and Test RMSE for logALS with varying number of features

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 5. 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.

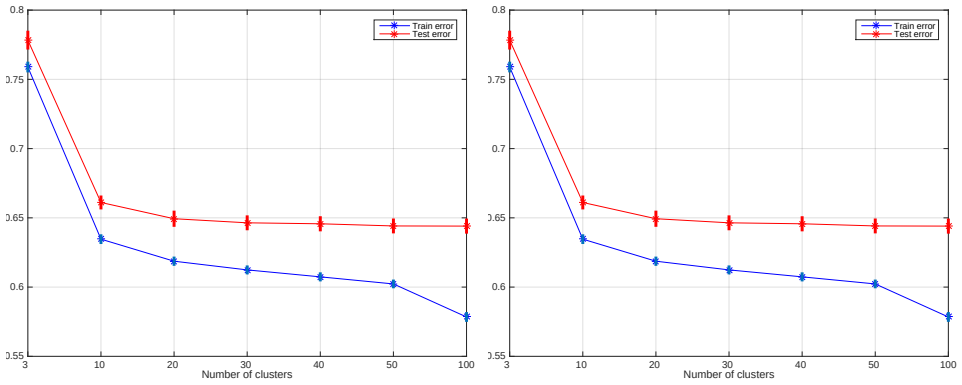
We noticed also the fact that sometimes the training error of Kmeans (reported by taking exp of data and RMSE) had small fluctuations and it was not always decreasig as the algorithm convergence properties would expect. This is due to the fact that Kmeans miminizes squared error but our cost is a little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

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.



(a) (b)

Figure 5: bla bla

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.

1.7 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.

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

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.

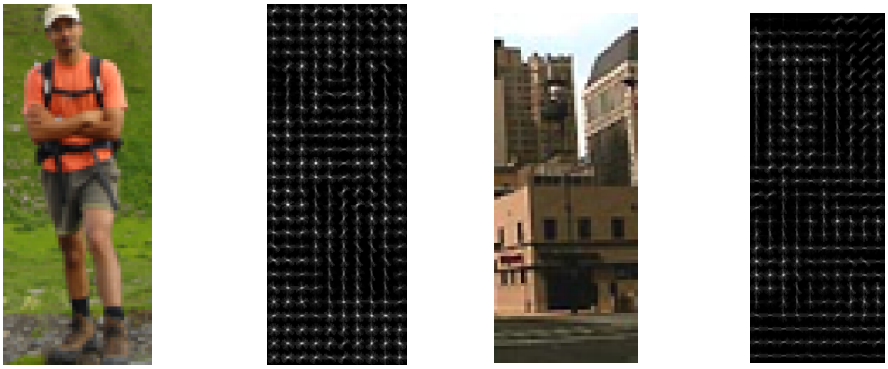
2 People Detection

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 form the training data that we will work on.

We notice that there are 7308 images without presence of people (negative images) and 1237 with (positive images). Fig6 presents a typical example from each category along with the corresponding feature descriptor.

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.



(a) Positive image. (b) Negative image.

Figure 6: Examples of positive and negative images and their corresponding feature descriptors.

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 the naive Bayes classifier, the logistic and penalized logistic regresion, the Support Vector Machine (SVM), the K-NN classifier, and the Neural Network classifiers. All of them are compared also with the random guess of the class in a later subsection of the report.

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 simple 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.

The results from the K-fold validation were not very satisfying, since the average TPR was very low (0.031). The reason that may be hidden behind that are the strong assumptions that we made in order to fit this model. Naive Bayes model assumes that the data that will process every time follow a specific distribution (in our case Gaussian), but also that the value of a particular feature is unrelated to the presence or absence of any other feature, given the variable of the class. In our case, this is possibly wrong, so the classifier cannot fit a appropriate model.

2.4 Logistic and Penalized Logistic Regression

Our purpose in this part was to fit the logistic regression and its penalized version to our data. The value for the parameter α that we used was 10^{-2}, since all of the values close to this performed well enough for our experiments. Moreover, regarding the value for λ in case of the penalized logistic regression, we have to mention that all of the values that were used performed the same. This is the reason that both of the method give the same results, with average TPR equal to 0.754.

During penalization the idea is to avoid the overfitting by imposing penalty on large fluctuations of the model parameters and reduce the influence of the outliers to our model. This is possibly the reason that both models perform equivalently. The data, as we mentioned before, do not have outliers and the fluctuations at the model parameters are avoided by the nature of our data.

2.5 Support Vector Machines

For the certain part of the experiment we used the LIBSVM 3.20 toolbox that includes SVM using various kernels. In our case, we performed analysis and K-fold validation on linear-kernel SVM, polynomial-kernel SVM with degrees 2 and 3, real-basis-kernel SVM and sigmoid-kernel SVM. In all of the cases we used the scores that were given by the program to compute the TPR and the FPR. A comparison of the various cases is given in Figure 7a.

As we can observe radial-basis-kernel SVM have significantly good performance. A fact that can be seen not only by the average TPR, which is 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 cases. The second best in performance is the polynomial SVM with degree 2, which has very similar results with the RBF case for FPR greater than $210e{-}3$. The rest of the classifier perform well for FPR greater than $210e{-}2$, with the exception of the linear case, which has quite good results for lower rates, too.

Moreover, in Table 2 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

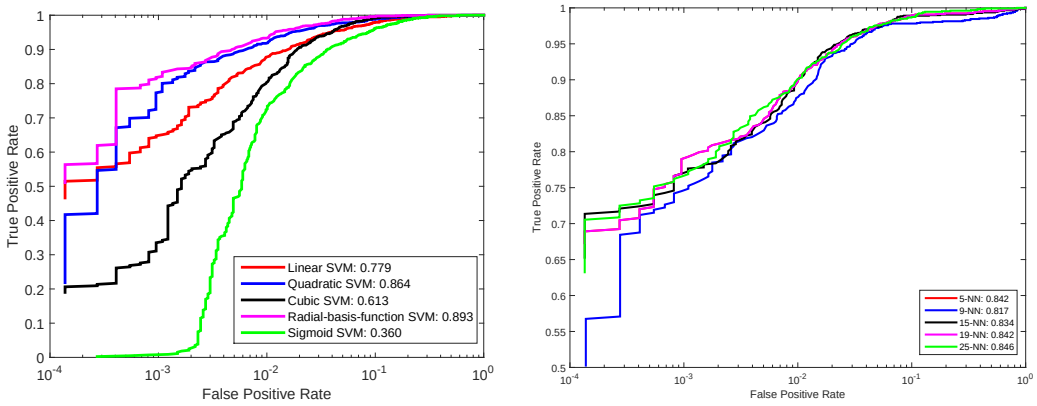
Table 2: Average percentage for the success rate for the various kernels of the SVM classifier, as they are given by the program of the LIBSVM 3.20 toolbox.

2.6 K-NN classifier

Using the K-NN classifier the most important parameter that we have to specify is the number of the neighbouring samples that will be used as a reference for the method so it will decide for any given sample the class that it belongs to. For our experiments we used a ready function, which we modified a lot in order to compute and give in output the scores of the prediction, except of the class labels. We varied the number of neighbours used among 5, 9, 15, 19 and 25. In addition, the euclidean distances were used as the way to specify the distances among the neighbouring samples.

Figure 7b presents the resulting ROC curves that were created using the scores got from the K-fold validation process. There we observe that all of the classifiers show very good performance,

especially, for FPR greater than 10^{-3}. Only the 9-NN classifier seems to perform a bit worse for FPR between 10^{-4} and 10^{-3}. These observations can be explained by the distribution of the data, where most probably they are distributed in the N-dimensional (N = 9360) space with such a way that their neighbours can specify, with high probability, the class that they belong to.



(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.

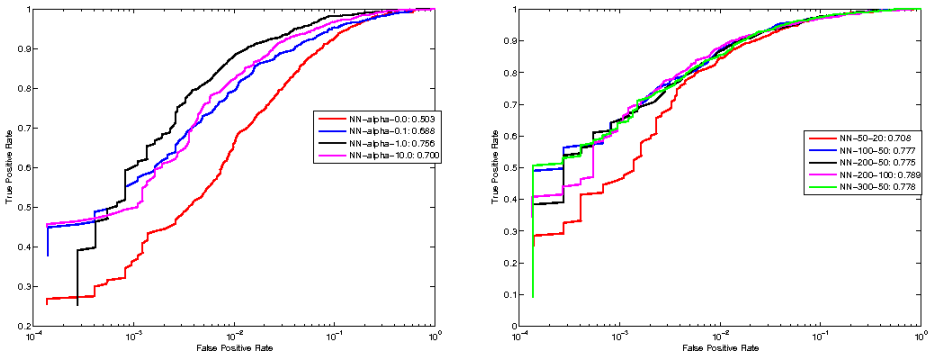
The parameters we considered for our problem were the number of epochs, the learning rate α (parameter in gradient descent update), the number of hidden layers and their size. We fixed the batch size to 50 and the number of epochs to 10, since we experimentally saw that the error starts to converge after 10 swipes through the data. The activation functions were also kept fixed to tanh 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 $[0.01, 0.1, 1, 10]$.

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 increase the model complexity by adding more neurons per layer or more hidden layers. The network $200 - 100$seemed to perform the best. One reason why $300 - 50$ did not perform better is either overfitting at the first layer or too few neurons at the second layer. Due to time and computationally contraint, we also run 200,100,50 and 300,100,50 and obtained the result in.

2.8 Convolutional Neural Network

We experimented a bit with convolutional neural networks. Our tests were unreasonable slow. We tried 6-12 and 5x5 kernels. This is a very small netwroks which performed bad. In order to have meaningful results we should have increased both parameters. We mention also that the library used did not offer the option of adding more hidden layers between the last convolutional and pool layer and the output. We believe this is a major drawback of the library, as the majority of setups used in literature have extra layers.



0.3

(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.

2.9 Comparison

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.