Inhaltsverzeichnis

1 Linear Regression and Gradient descent 2
   1.1 Polynomial Regression [6.5 points] .......................... 2
       1.1.1 Hints ............................................. 3
   1.2 Gradient Descent [6 points] ............................... 3
       1.2.1 Hints ............................................. 3

2 Neural Networks 3
   2.1 Simple Regression with Neural Networks[4 points] ........... 3
       2.1.1 Hints ............................................. 4
   2.2 Face Recognition with Neural Networks [8.5 points] .......... 4
       2.2.1 Pose Recognition .................................. 4
       2.2.2 Face Recognition .................................. 4
       2.2.3 Hints ............................................. 4
       2.2.4 Remarks .......................................... 5

3 Neural Networks and Regularization 5
   3.1 Simple Regression with Neural Networks (continued)[4 points] 5
       3.1.1 Hints ............................................. 5
   3.2 Overfitting with Real-World Data [8.5 points] ................ 6
       3.2.1 Hints ............................................. 6
       3.2.2 Remarks .......................................... 6

4 Decision Trees and Support Vector Machines 7
   4.1 Decision Trees [5 points] ................................ 7
       4.1.1 Hints ............................................. 7
   4.2 Support Vector Machines [7.5 points] .......................... 8
       4.2.1 Hints ............................................. 8

5 VC-Dimension and Proofs 8
   5.1 Threshold circuits [8.5 points] .............................. 8
   5.2 VC-dimension [4 points] ................................... 9

6 Gaussian Statistics and Unsupervised Learning 9

7 Statistical Pattern Recognition 9

8 Unsupervised Learning 10
   8.1 Clustering using the K-means algorithm ..................... 10
   8.2 Clustering using the EM algorithm ........................ 11
9 Principal Component Analysis (PCA) and Independent Component Analysis (ICA)  

10 Hidden Markov Models  

11 Mixtures of Gaussians and HMMs  

12 Adaptive Filters  

1 Linear Regression and Gradient descent  

[Points: 12.5; Issued: 2008/03/14; Deadline: 2008/05/05; Tutor: Sabine Sternig\(^1\); Infohour: 2008/04/29, 13:00-14:00, Seminarraum Infeldgasse 16b, 1. Stock; Einsichtnahme: 2008/05/16, 15:30-16:30, HS i11;]

1.1 Polynomial Regression [6.5 points]  

Consider a 10-degree polynomial model. Use an additive model with 11 basis functions \( \phi_k(x) = x^k \)  
\((k = 0 \ldots 10)\) and the following error function for your training examples  

\[
E(w) = \sum_{i=1}^{N} (y_i - \sum_{k=0}^{10} \phi_k(x_i)w_k)^2 + \alpha \sum_{k=0}^{10} w_k^2
\]

- Write the error function in matrix form. Explicitly state the dimensions of the vectors and matrices.  
- Derivate a closed form solution for the optimal weight vector. (Hint: Use the identity \( \alpha w = \alpha I w \), \( I \) being the identity matrix.)  
- Write a matlab script which implements your learning rule. Use the following data set\(^2\), which contains the training data (input: \( x_{\text{train}} \), output: \( y_{\text{train}} \)) and the data for testing (input: \( x_{\text{test}} \), output: \( y_{\text{test}} \)).  
- Train your model with the trainings data using \( \alpha \) values of 0:0.01:10  
- Plot the mean squared error of the training and of the test set for the given \( \alpha \)s.  
- Plot the learned functions for \( \alpha = 0 \), and \( \alpha = 10.0 \) the best \( \alpha \) for the error on the test set. Interpret your results.  
- Plot the mean absolute weight values for the given \( \alpha \) (use a semilogy plot for better illustration).  
- Interpret your results, what is the purpose of \( \alpha \)?

\(^1\)mailto:sabinest@box.tugraz.at  
\(^2\)http://www.igi.tugraz.at/lehre/CI/homework/data/homework1.mat
1.1.1 Hints

- For a single training example \( x \), the basis functions can be easily created by \( x.^(0:10) \)

1.2 Gradient Descent [6 points]

Consider the following feedforward neural network with a 1-dimensional input, \( K \) outputs an \( M \) hidden units, where the \( k \)th output is given by:

\[
y_k(x) = \sum_{j=1}^{M} w_{kj} f((x - \mu_j)^2),
\]

where \( f \) is an arbitrary function. Derive a gradient descent learning rule for the weights \( w_{kj} \) and \( \mu_j \) which minimizes the mean squared error (mse) of a single example \( (x, b) \).

1.2.1 Hints

- Use \( f' \) as the derivation of function \( f \)
- The chain rule is your friend... use it!
- Please state the whole weight update rule, the gradient alone is not sufficient.

2 Neural Networks

[Points: 12.5; Issued: 2008/03/20; Deadline: 2008/05/16; Tutor: Robert Peharz\(^3\); Infohour: 2008/05/09, 15:30-16:30, HS i11; Einsichtnahme: 2008/05/30, 15:30-16:30, HS i11;]

2.1 Simple Regression with Neural Networks[4 points]

In this task a simple 1-dimensional function should be learned with feed-forward neural networks. Use the same data set\(^4\) as for homework 1.

- Train a neural network with \( n = [1, 2, 3, 4, 6, 8] \) neurons. Use the training algorithm 'trainscg', train for 700 epochs.
- Plot the mean squared error of the training and of the test set for the given number of neurons. For the test set, plot the mean squared error (mse) after training \( (tmse_{end}) \) and the minimum mse during training \( (tmse_{min}) \).
- Interpret your results. Why is there a difference between \( tmse_{end} \) and \( tmse_{min} \)? What is the best value of \( n \)?
- Plot the learned functions for \( n = 1, n = 2 \) and \( n = 10 \). Interpret your results, refer to results from the previous plots!
- Compare the results to homework 1. Is there any connection of \( n \) and \( \alpha \)? [2 Extra-Points]

\(^3\)mailto:peharz@sebox.tugraz.at
\(^4\)http://www.igi.tugraz.at/lehre/CI/homework/data/homework1.mat
2.1.1 Hints

- Normalize your input data using `mapstd` (in older Matlab versions (≥ 7.5) this function is called `prestd`)
- You can easily use the performance structure returned by the `train` function to get the error on the training and on the test set.

2.2 Face Recognition with Neural Networks [8.5 points]

In this task you are asked to work with the dataset `faces.mat` which contains face images. The dataset contains images of different persons, with different pose (straight/left/right/up), with/without sunglasses and showing different emotions. Download the Matlab dataset. It contains 2 datasets: `dataset1 (input1, target1)` with 60 data points and `dataset2 (input2, target2)` with 564 data points. The `target` matrices contain the class information. The first column codes the person, the second column the pose, the third column the emotion and the last column indicates whether the person is wearing sunglasses. In `template_faces.m` you can find a script for training a sunglasses recognizer. This script can be used as template. Additionally you need to download the file `confmat.m` which is needed to calculate the confusion matrix.

2.2.1 Pose Recognition

- Train a 2 layer feed-forward neural network with 6 hidden units for pose recognition. Use `dataset2` for training, `trainscg` as training algorithm and train for 300 epochs. Do not use any test set.
- State the confusion matrix on the training set. Are there any poses which can be better separated than others?
- Plot the weights of the hidden layer for every hidden unit. Can you find particular regions of the images which get more weights than others? Do particular units seem to be tuned to particular features of some sort?

2.2.2 Face Recognition

- Train a 2 layer feed-forward neural network with 20 hidden units for recognizing the individuals. Use `dataset1` for training, `trainscg` as training algorithm and train for 1000 epochs. Use `dataset2` as test set.
- Repeat the process 10 times starting from a different initial weight vector. Plot the histogram for the resulting mean squared error (mse) on the training and on the test set.
- Interpret your results! Explain the variance in the results.
- Use the best network (with minimal mse on the test set) to calculate the confusion matrix for the test set and the mean classification error (not the mse!) on the test set. Plot a few missclassified images. Do they have anything in common?

2.2.3 Hints

- Normalize your input data using `mapstd` (in older Matlab versions (≥ 7.5) this function is called `prestd`).

---

5 http://www.igi.tugraz.at/lehre/CI/homework/data/faces.mat
6 http://www.igi.tugraz.at/lehre/CI/homework/data/faces_template.m
7 http://www.igi.tugraz.at/lehre/CI/homework/data/confmat.m
• In the template script you can find code for plotting an image and plotting the weights of a hidden neuron.
• Be aware that the template script only covers the 2 class classification case!
• Use the functions `full` and `ind2vec` to get from the standard class coding to a 1 out of n coding.

2.2.4 Remarks
• Hand-in your matlab code as print-outs (no emails !!).
• Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortlessly.
• Use your matrikel number to initialize the state of the random number generators.

3 Neural Networks and Regularization

3.1 Simple Regression with Neural Networks (continued)[4 points]

Similiar to homework 2.1, a simple 1-dimensional function should be learned with feed-forward neural networks. Use the same data set9 as for homework 1.

• Train a neural network with \( n = 10 \) neurons. Use the training algorithm `trainscg`, train for 700 epochs. Use the regularized error function `msereg`. Use different regularization factors (\( \alpha \) resp. `net.performParam.ratio` in matlab) of \( \alpha = [0.9, 0.95, 0.975, 0.99, 0.995, 1.0] \);
• Plot the mean squared error of the training and of the test set for the given regularization factors.
• Interpret your results. What is the best value of \( \alpha \)? Also interpret your results with homework 2.1. Is the appropriate selection of the number of hidden neurons, early stopping or the regularized error function the best choice to avoid overfitting in this example? Explain your choice?[1 Extra Point]

3.1.1 Hints
• Normalize your input data using `mapstd`
• This time you CAN NOT use the performance structure returned by the `train` function because it returns the regularized error function and not the mse. Use the `mse` function instead.

8`mailto:caleb@sbox.tugraz.at`
9`http://www.igi.tugraz.at/lehre/CI/homework/data/homework1.mat`
3.2 Overfitting with Real-World Data [8.5 points]

In this homework you are supposed to analyse different overfitting avoidance mechanism with neural networks. Use the housing.mat\(^{10}\) dataset which contains data about the price of houses in boston. The task is to predict the price of unseen houses (regression task). For a detailed description of the dataset see housing_description.txt\(^{11}\). You can use the script housing_template.m\(^{12}\) as template.

- Split the dataset randomly (a useful command is randperm) in a training set \(D\) (75%) and a test set \(T\) (25%).

- Train a neural network with \(n = [2, 5, 10, 20, 40, 60]\) neurons. Use the training algorithm trainscg and learn for 500 epochs.

- Plot the mean squared error of the training and the test set for the given number of neurons. For the test set, plot the mean squared error (mse) after training (\(tmse_{end}\)) and the minimum mse during training (\(tmse_{min}\)). For which number of neurons can we observe (with standard training, i.e. for \(tmse_{end}\)) underfitting and for which overfitting?

- What is the best number of hidden neurons for standard training (\(tmse_{end}\)) and what for early stopping (\(tmse_{min}\))? Is there a difference between those two numbers? If yes, why?

- Train a neural network with 60 hidden neurons with the regularized error function msereg. Use the training algorithm trainscg and learn for 500 epochs. Use the following \(\alpha\) values: \(\alpha = [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1.0]\);

- Plot the mean squared error of the training and the test set for the given \(\alpha\). What is the best \(\alpha\)? Good working \(\alpha\) values for this example are significantly different from example 3.1. Why? For which \(\alpha\) can we observe underfitting and for which overfitting?

- Compare the results to standard training and early stopping with varying number of hidden units. Which OF avoidance method would you prefer? Explain your choice!

- Repeat the whole experiments using a 4-fold cross-validation instead of the testset to estimate the true error. Create the same plots and compare them. Which results are more reliable? Does your choice of the optimal number of hidden neurons and the optimal \(\alpha\) change?

3.2.1 Hints

- Normalize the data using mapstd.

- You can find the source code for splitting the data in a training and a test set in the template.

3.2.2 Remarks

- Present your results clearly, structured and legible. Document them in such a way that anybody can easily reproduce them.

- Please hand in the print out of the Matlab program you have used (no emails!).

---

\(^{10}\)http://www.igi.tugraz.at/lehre/CI/homework/data/housing.mat

\(^{11}\)http://www.igi.tugraz.at/lehre/CI/homework/data/housing_description.txt

\(^{12}\)http://www.igi.tugraz.at/lehre/CI/homework/data/housing_template.m
4 Decision Trees and Support Vector Machines

[Points: 12.5; Issued: 2008/03/20; Deadline: 2008/05/26; Tutor: Martin Stadler13; Infohour: 2008/05/16, 15:30-16:30, HS i11; Einsichtnahme: 2008/06/06, 15:30-16:30, HS i11;]

4.1 Decision Trees [5 points]

In this example you are asked to implement the entropy impurity and also the gain ratio impurity calculation in order to build a decision tree. Download and unzip the decision tree framework for matlab decisiontrees.zip14. This framework consists of several functions for building a decision tree from a given data set, classify new data point with a decision tree and several other evaluation functions. Please refer to the trees_template.m file in order to see how these functions work.

The function getScore calculates a score for splitting a certain attribute. As input, it gets the attribute values of dimension $j$ of all training examples in $L_v$ and also the class labels of all examples in $L_v$. The attribute values can be any integer value, the class labels are either 1 or 2.

At each new node, the function getScore is called for each dimension $j$. The dimension with the highest score is used for splitting. By now, the getScore function only returns random values, the correct implementation of these functions is your task.

• Use the dataset treedataset1.csv. Build the tree with the already existing random getScore function and determine the depth, the number of leafes and the error of the tree by performing a 10 fold cross validation.

• Implement the information gain (with the help of the entropy impurity) in the getScore function. Again build the tree with treedataset1.csv and evaluate the depth and the number of leaves of the tree. Estimate the true error of the tree by performing a 10 fold cross validation. Compare it to the previous results. Explain the differences in the cross validation error.

• Use the dataset treedataset2.csv. It is basically the same dataset with an additional attribute added. The new attribute has 50 random nominal values. Again build your tree with the entropy impurity score function and determine depth, number of leaves and the cross-validation error. Can you observe a change in the cross-validation error? If yes, why?

• Implement the gain ratio impurity in the getScore function. Build the tree with the dataset treedataset2.csv and evaluate the depth, the number of leaves of the tree and the cross validation error. Compare it to the previous result.

4.1.1 Hints

• Be careful when calculating the product $p \log(p)$ in Matlab when $p = 0$. Matlab returns NaN in this case.

• The function em unique might be useful to determine all possible attribute values.

4.2 Support Vector Machines [7.5 points]

In this homework you are supposed to evaluate different parameter settings for the Support Vector Machine (svm). Download the svm library15 for matlab and follow the install instructions (install.txt). Consult the files demsvm1.m or svm_template.m to get an overview of the basic methods of the svm library.

13mailto: martin.stadler@tugraz.at
14http://www.igi.tugraz.at/lehre/CI/homework/data/decisiontrees.zip
15http://www.igi.tugraz.at/lehre/CI/homework/data/svm.zip
• The file homework4.mat contains a training set \((x_{\text{train}}, t_{\text{train}})\) and a test set \((x_{\text{test}}, t_{\text{test}})\). The file is already included in the svm directory.

• Train several svms with different polynomial kernels. Use \(p = [1, 2, 3, 4, 5]\) for the degree of the kernel. Plot the training points, the decision boundary and the support vectors for all settings of \(p\) (see template). Compare the plots. Also plot the error on the test set and on the training set for increasing \(p\) values. Which \(p\) is the best setting?

• Train several svms with different rbf kernels (bandwidth set to \([0.1, 10, 100]\)) and \(C\) set to 10. Plot the training points, the decision boundary and the support vectors (see template) for all three settings. Interpret the plots.

• Again train svms with different rbf kernels. This time set the bandwidths to \(\text{logspace}(-1, 2, 25)\) and compute the error on the test set for each bandwidth. Repeat the experiment for \(C = [0.1, 1.0, 10.0, 50.0]\). Plot the test set error versus the rbf kernel bandwidth for each setting of \(C\) (i.e. for each \(C\) setting you get an individual curve), use a semilogz plot to get to a good scaling of the curves. Interpret your results, can you find any relation between good \(C\) values and good rbf bandwidths? Can you find any explanation for this relationship?

• Train a feed-forward neural network with 20 hidden neurons using weight decay. Use a learning algorithm of your choice, try different regularization constants \((\text{net.performParam.ratio})\). Can you achieve a comparable performance than with svms?

4.2.1 Hints
• The class labels used for the svm are 1 and \(-1\). Keep that in mind when training the neural network.

5 VC-Dimension and Proofs
[Points: 12.5; Issued: 2008/05/09; Deadline: 2008/05/30; Tutor: Gerhard Neumann\(^{16}\); Infohour: 2008/05/23, 15:30-16:30, HS i11; Einsichtnahme: 2008/06/13, 15:30-16:30, HS i11;]

5.1 Threshold circuits [8.5 points]

Prove that each Boolean function can be computed by a threshold circuit (Schwellenschaltkreis) of depth 2.

Remarks
• You can use the fact (without proving it) that there exists for each Boolean function \(f: \{0, 1\}^n \rightarrow \{0, 1\}\) a Boolean formula in disjunctive normal form (DNF); see definition 1.1.5 of “Grundbegriffe der Aussagenlogik”\(^{17}\) for the precise definition of the DNF.

• Hence you just have to prove that there exists for each Boolean formula in DNF a circuit of threshold gates (of depth 2) which outputs for each binary input vector 1 if the DNF formula is true and outputs 0 if the DNF formulae is false for the corresponding truth assignment.

\(^{16}\texttt{mailto:neumann@igi.tugraz.ac.at}\)
\(^{17}\texttt{http://www.igi.tugraz.at/lehre/CI/library/aussagenlogik.pdf}\)
5.2 VC-dimension [4 points]

• Consider the case of the hypothesis class of axis parallel rectangles. The VC-dimension of a single rectangle is 4. Find the best lower bound if you can use 2 rectangles instead of one!

• Consider the case of the hypothesis class of a single, axis parallel rectangle, where you can additionally choose whether to classify class 1 inside or outside the rectangle. What's the best lower bound for the VC-dimension you can prove?

6 Gaussian Statistics and Unsupervised Learning

Go through the tutorial “Gaussian Statistics and Unsupervised Learning” and download the accompanying MATLAB programs and data. The tasks issued in this homework are a subset of the tasks in the tutorial.

You can use a printout of this homework designation as a basis for your elaboration and fill in your results and answers. Add printouts of all requested figures. Do not forget to add your name and “Matr.Nr.” on top.

Please add the printout of your MATLAB scripts used for producing the results to your report.

7 Statistical Pattern Recognition

1. Load data from file “pendigit.mat”. Supposing that the whole database covers adequately an imaginary language made up only of /a/’s, /e/’s, /i/’s, /o/’s and /y/’s, compute the probability \( P(q_k) \) of each class \( q_k \), \( k \in \{/a/, /e/, /i/, /o/, /y/\} \). Which is the most common and which is the least common phoneme in the language?

\[
P(q_a) = P(q_e) = P(q_i) = P(q_o) = P(q_y) =
\]

Most common phoneme:

Least common phoneme:

2. Plot the clouds of observed feature points \( x = [F_1, F_2]^T \) for all vowels:

\[
>> \text{plotvow}
\]

Train the Gaussian models corresponding to each class (use directly the \texttt{mean} and \texttt{cov} commands). Then compute and plot the Gaussian models:

\[
>> \text{mu}_a = \text{mean}(a);
\]

\[
>> \text{sigma}_a = \text{cov}(a);
\]

\[
>> \text{plotgaus(mu}_a, \text{sigma}_a, [0 1 1]);
\]

Print a plot showing the pdfs of the Gaussian models for the five phoneme classes of our imaginary language, label each Gaussian with the according class and the axes, and append the plot to this homework.

\[\text{mailto:mwiese@sbox.tugraz.at}\]
3. Now, we have modeled each phoneme class with a Gaussian pdf (by computing means and variances), we know the probability \( P(q_k) \) of each class in the imaginary language, and we assume that the speech feature vectors \( \mathbf{x} \) (as opposed to the phoneme classes) are equiprobable.

What is the most probable class \( q_k \) for the speech feature points \( \mathbf{x}_i = [F_1, F_2]^T + [A, A]^T \) given in the table below? Symbol \( A \) denotes the last two digits of your Matrikelnummer. Use the Bayesian discriminant function \( f_k(\mathbf{x}) = \log p(\mathbf{x} | q_k, \Theta) + \log P(q_k | \Theta) \), proportional to the log of the posterior probability of \( \mathbf{x} \) belonging to class \( q_k \).

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \mathbf{x}_i = [F_1, F_2]^T )</th>
<th>( f_{/i/}(\mathbf{x}_i) )</th>
<th>( f_{/e/}(\mathbf{x}_i) )</th>
<th>( f_{/i/}(\mathbf{x}_i) )</th>
<th>( f_{/e/}(\mathbf{x}_i) )</th>
<th>Most prob. class ( q_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>([200 + A, 1500 + A]^T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>([400 + A, 1000 + A]^T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>([530 + A, 1000 + A]^T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>([600 + A, 1000 + A]^T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>([670 + A, 1000 + A]^T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>([700 + A, 2000 + A]^T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The iso-likelihood lines for the Gaussian pdfs \( \mathcal{N}(\mu_{/i/}, \Sigma_{/i/}) \) and \( \mathcal{N}(\mu_{/e/}, \Sigma_{/e/}) \), which we used before to model the class \( /i/ \) and the class \( /e/ \), are plotted in Figure 1 of the tutorial. On a second graph, the iso-likelihood lines for \( \mathcal{N}(\mu_{/i/}, \Sigma_{/e/}) \) and \( \mathcal{N}(\mu_{/e/}, \Sigma_{/e/}) \) (two pdfs with the same covariance matrix \( \Sigma_{/e/} \)) are represented. On the figures, use a colored pen to join the intersections of the level lines that correspond to equal likelihoods. (If the parameters of your Gaussian models for \( /e/ \) and \( /i/ \) are on the workspace, you can also use \texttt{isosurf} in \texttt{MATLAB} to create a color plot.)

- What is the nature of the surface that separates class \( /i/ \) from class \( /e/ \) when the two models have different covariance matrices? Can you explain the reason for this form?
- What is the nature of the surface that separates class \( /i/ \) from class \( /e/ \) when the two models have the same covariance matrices? Why is it different from the previous discriminant surface?

### 8 Unsupervised Learning

We will now use two algorithms for unsupervised learning to cluster the feature vectors \( \mathbf{x} \) for all phonemes of the imaginary language, i.e., we try to automatically find the distributions of the feature vectors for the 5 phoneme classes.

#### 8.1 Clustering using the \( K \)-means algorithm

Launch the \( K \)-means explorer \texttt{kmeans} with the data sample \texttt{allvow} (stored in file \texttt{pendigit.mat}) which gathers the feature vectors for all phonemes of our imaginary language in one matrix. Do several runs with each of the following different initializations of the algorithm:

1. 5 initial clusters determined according to the default heuristic;
2. some initial \texttt{MEANS} values equal to some data points;
3. some initial MEANS values equal to \( \{ \mu_a, \mu_e, \mu_i, \mu_o, \mu_y \} \).

Iterate the algorithm until its convergence. Observe the evolution of the cluster centers, of the data-points attribution chart and of the total squared Euclidean distance. (If you enable button “Zoom in” in the “Tools” menu it is possible to zoom the plots: left click inside the axes to zoom 2\times centered on the point under the mouse; right click to zoom out; click and drag to zoom into an area; double click to reset the figure to the original size.) Observe the mean values found after the convergence of the algorithm.

**Example:**

```matlab
>> kmeans(allvow,5);
```

- or -

```matlab
>> means = {mu_a, mu_e, mu_i, mu_o, mu_y};
>> kmeans(allvow,means);
```

Enlarge the window, then push the buttons, zoom etc. After the convergence, use:

```matlab
>> for k=1:5, disp(kmeans_result_means{k}); end
```

to see the resulting means.

**Answer the following questions:**

1. Does the final solution of the \( K \)-means algorithm depend on the initialization (Advice: Use e.g. 10 clusters)?

2. Describe the evolution of the total squared Euclidean distance.

3. What is the nature of the discriminant surfaces corresponding to a minimum Euclidean distance classification scheme?

4. Is the algorithm suitable for fitting Gaussian clusters? Why/why not?

### 8.2 Clustering using the EM algorithm

Launch the EM explorer emalgo with the same dataset allvow. Do several runs with different initializations of the algorithm:

1. 5 clusters determined according to the default heuristic;

2. some initial MEANS values equal to some data points, and some random VARS values (e.g., cov(allvow) for all the classes);

3. the initial MEANS and VARS values found by the \( k \)-means algorithm.

4. the initial MEANS values equal to \( \{ \mu_a, \mu_e, \mu_i, \mu_o, \mu_y \} \),
   initial VARS values equal to \( \{ \Sigma_a, \Sigma_e, \Sigma_i, \Sigma_o, \Sigma_y \} \),
   and initial PRIORS values equal to \( \{ P(q_a), P(q_e), P(q_i), P(q_o), P(q_y) \} \);

5. some initial MEANS and VARS values chosen by yourself.

Iterate the algorithm until the total likelihood reaches an asymptotic convergence. Observe the evolution of the clusters and of the total likelihood curve. (In the EM case, the data points attribution chart is not given because all data points participate in the update of each cluster.) Observe the mean, variance and prior values found after the convergence of the algorithm. Compare them with the real mean and covariance values of the vowels. Describe your observations.

(If you have time, also increase the number of clusters and play again with the algorithm.)
Example:
```matlab
>> emalgo(allvow,5);
  or –
>> means = {mu_a,mu_e,mu_i,mu_o,mu_y};
>> vars = {sigma_a,sigma_e,sigma_i,sigma_o,sigma_y};
>> emalgo(allvow,means,vars);
```

Enlarge the window, then push the buttons, zoom etc. After convergence, use:
```matlab
>> for k=1:5, disp(emalgo_result_means{k}); end
>> for k=1:5, disp(emalgo_result_vars{k}); end
>> for k=1:5, disp(emalgo_result_priors(k)); end
```
to see the resulting means, variances and priors.

Questions:

1. Does the final solution of the EM algorithm depend on the initialization of the algorithm?
2. Describe the evolution of the total likelihood. Is it (always) monotonic?
3. In terms of optimization of the likelihood, what does the final solution correspond to?
4. Is the algorithm suitable for fitting Gaussian clusters? Why/why not?

9 Principal Component Analysis (PCA) and Independent Component Analysis (ICA)

[Points: 12.5; Issued: 2008/05/23; Deadline: 2008/06/13; Tutor: Johannes Schweighofer\(^\text{19}\); Infoblau: 2008/05/30, 15:30-16:30, HS i11; Einsichtnahme: 2008/06/20, 15:30-16:30, HS i11;]

Please read the tutorial about PCA, ICA, Blind Source Separation carefully before you start the assignment. Accompanying MATLAB programs and data can be downloaded from the homepage.

Do not forget to add your name and “Matr.Nr.”.

Produce all the plots in the report. Label the axis of your plots. Give explanations to your plots.

Add to the end of the report the MATLAB script you programmed for producing all the results.

1. Problem: Generate 3-dimensional uniformly distributed data with \( N = 2000 \) samples using the command `rand()`.

   Multiply the Data using the following mixing matrix

   \[
   \begin{bmatrix}
   2 & 3 & 1 \\
   2 & 1 & 0.5 \\
   1 & 1 & 1
   \end{bmatrix}
   \]

   Visualize the data in the 3-dimensional domain using `plot3()`.

   2. Problem: Generate 4 signals using the following commands:

   ```matlab
   mailto:johannes.schweighofer@student.tugraz.at
   ```
N=500
v=[0:N-1];
s=[];
s(1,:)=sin(v/2); %sinusoid
s(2,:)=((rem(v,23)-11)/9).^5; %funny curve
s(3,:)=((rem(v,27)-13)/9); %saw-tooth
s(4,:)=rand(1,N); %random noise

Normalize the signals s so that they have zero mean and unit variance by using the command mean() and std(). Visualize the four signals.
Mix the 4 signals by an arbitrary mixing Matrix \( A \) (rand(4)) to get the sensor signals received at the 4 microphones \( x = As \).
Visualize the generated and the mixed signals. Now apply ICA to recover the original source signals \( S \) by using the provided command fastica(). Plot the resulting signals after applying ICA. Interpret your results. What happens when you perform ICA? Can you establish the original ordering and sign of the signals after applying ICA? Why? Is ICA working for two Gaussian distributed signals \( s \)? Why?

3. Problem: Apply ICA similarly as in the previous task using two sound files. You can use the command wavread() to load the wav files sourceX.wav. Select arbitrarily two different sound files for the experiments. You can play the files using sound(). Again plot the loaded, mixed and unmixed signals. Is ICA separating the mixed signals well? Listen to the unmixed signals after applying ICA.

4. Problem: Apply PCA to the previous task using two sound files. Is PCA capable to separate the mixed signal again. Listen and plot the PCA transformed signals? Explain your observations.

10 Hidden Markov Models

[Points: 12.5; Issued: 2008/05/30; Deadline: 2008/06/13; Tutor: Susanne Rexeis\(^{20}\); Infhour: 2008/06/06, 15:30-16:30, HS i11; Einsichtnahme: 2008/06/27, 15:30-16:30, HS i11;]

Go through the tutorial “Hidden Markov Models” and download the accompanying MATLAB programs and data. The tasks issued in this homework are a subset of the tasks in the tutorial.
You can use a printout of this homework designation as a basis for your elaboration, and fill in your results in the tables. Add sheets of paper with descriptions of your observations and printouts of MATLAB figures. Add to the end of the report the MATLAB script and your Viterbi decoding function vit_ghmm you programmed for producing all the results.
Additionally, you should send the matlab code of the viterbi function for generating the results to the tutor (susanne.rexeis@student.tugraz.at) with the subject EW. Please provide your Name and Matrikelnummer in the scripts and in the body of the email.

1. Load the Hidden Markov Models (HMMs) \( \Theta_1, \ldots, \Theta_4 \), and make a sketch of each of the models with states and transition probabilities. The parameters of the Markov models and of the Gaussian emission pdfs are stored in the file data.mat. Each HMM \( \Theta_i \) is stored as an object, e.g., hmi1, with fields hmi1.trans, hmi1.pi, hmi1.means, and hmi1.vars. The trans field contains the transition matrix \( A \), and the pi field the prior probability vector \( \pi \). The means field contains a matrix composed of mean vectors of the Gaussian emission pdfs, where each column of the matrix corresponds to one state of the HMM (to access the mean vector \( \mu \) of, e.g., the second state from hmi1 use: hmi1.means(:,2). The vars field contains

20mailto:susanne.rexeis@student.tugraz.at
a 3 dimensional array of covariance matrices, where the third dimension corresponds to the state (e.g., to access the covariance matrix \( \Sigma \) of state 1 use \( \text{hmm1.vars(:,:,1)} \)).

2. Generate samples from the HMMs \( \text{hmm1} \), \( \text{hmm2} \), \( \text{hmm3} \), and \( \text{hmm4} \) and plot them with \text{plotseq} and \text{plotseq2}. In the resulting plots, the obtained sequences are represented by a yellow line where each point is overlaid with a colored dot. The different colors of the dots indicate the state from which a particular sample has been drawn.

\[
\text{>> % Example: generate a sequence of length } T \text{ (e.g. } 80) \text{ from HMM1}
\text{>> [X,ST] = sample_ghmm(hmm1,T)}
\text{>> plotseq(X,ST) % View of both dimensions as separate sequences}
\text{>> plotseq2(X,ST,hmm) % 2D view with location of Gaussian states}
\]

Draw several sequences for each HMM and compare. Compare the MATLAB figures with your sketch of the models and add (some of) them to your homework elaboration. Moreover, explain: What is the effect of the different transition matrices of the HMMs on the sequences obtained? Hence, what is the role of the transition probabilities in the HMM?

3. Pattern recognition with HMM's: Classify the sequences \( X_1, X_2, X_3, X_4 \), given in the file \( \text{Xdata.mat} \), in a maximum likelihood sense with respect to the four Markov models \( \Theta_1, \Theta_2, \Theta_3, \text{ and } \Theta_4 \). Use the function \text{loglik_ghmm} to compute the log-likelihood \( \log p(X_i|\Theta_j) \) of a sequence \( X_i \) with respect to a HMM \( \Theta_j \). Store the results in a matrix (they will be used in the next section).

\[
\text{>> load Xdata}
\text{>> % Example:}
\text{>> logLike(1,1) = loglik_ghmm(X1,hmm1)}
\text{>> logLike(1,2) = loglik_ghmm(X1,hmm2)}
\text{...}
\text{>> logLike(i,j) = loglik_ghmm(Xi,hmmj)}
\text{...}
\]

Filling the \text{logLike} matrix can be done automatically with the help of loops:

\[
\text{>> for i=1:4,}
\text{for j=1:4,}
\text{stri = num2str(i);}
\text{strj = num2str(j);}
\text{eval([' 'logLike(',' , stri , ',' , strj , ')'=...}
\text{loglik_ghmm(X' , stri , ',' ,hmm' , strj , ');']); ]};
\text{end;}
\text{end;}
\]

You can find the maximum of each row in the matrix with the MATLAB function \text{max}:

\[
\text{>> for i=1:4;}
\text{[v,index] = max(logLike(i,:));}
\text{disp([''X',num2str(i),' -> HMM',num2str(index)]);};
\text{end}
\]
4. Viterbi decoder: Write a MATLAB function \([\text{loglik,path} = \text{vit}_\text{ghmm}(\text{data},\text{HMM})]\) to implement the Viterbi decoding algorithm to find the most likely state sequence \(Q\) for a given observation sequence \(X_i\) for HMMs with Gaussian emission probabilities. Use the function \(\text{mk}_\text{ghmm}_\text{obs}_\text{lik}\) to calculate the observation probabilities (Gaussian) for each state and time step. Please perform the calculations in the log domain, where the multiplications of the probabilities for the parameters \(\delta\) and \(\psi\) become additions. So you can prevent numerical issues for long sequences.

\[
\begin{align*}
[\text{loglik,path} = \text{vit}_\text{ghmm}(\text{data},\text{HMM})] \\
% Compute the path and the log likelihood of a given model and observation sequence \\
% INPUT \\
% data ... containing a sequence of observation \\
% size(data)=[Number of features of each obs., length of sequence] \\
% HMM is an object containing \\
% HMM.trans = state transition probability matrix; \\
% HMM.pi = prior probability vector; \\
% HMM.means = mean vectors of Gaussian emission pdf for each state; \\
% HMM.vars = covariance matrices of Gaussian em. pdf for each state; \\
% OUTPUT \\
% loglik ... log likelihood of the most likely path for the data \\
% path ... most likely path
\end{align*}
\]

5. Use your function \(\text{vit}_\text{ghmm}\) to compute the most likely paths for the sequences \(X_1,\ldots,X_4\) with respect to each model \(\Theta_1,\ldots,\Theta_4\). Also compute the log-likelihoods \(\log p^*(X_i|\Theta_j)\) along the most likely paths found by the Viterbi decoder. Note down your results below. Compare with the log-likelihoods \(\log p(X_i|\Theta_j)\) obtained in the previous section with the function \(\text{loglik}_\text{ghmm}(\ldots)\):

\[
\text{diffL} = \text{logLike-logLikeViterbi}
\]

Log-likelihoods along the best path:

| Sequence | \(\log p^*(X_i|\Theta_1)\) | \(\log p^*(X_i|\Theta_2)\) | \(\log p^*(X_i|\Theta_3)\) | \(\log p^*(X_i|\Theta_4)\) | Most likely model |
|----------|-------------------------|-------------------------|-------------------------|-------------------------|------------------|
| \(X_1\)  |                         |                         |                         |                         |                  |
| \(X_2\)  |                         |                         |                         |                         |                  |
| \(X_3\)  |                         |                         |                         |                         |                  |
| \(X_4\)  |                         |                         |                         |                         |                  |

Difference between log-likelihoods of a sequence given the model and log-likelihoods along the best path found by the Viterbi algorithm, \(\log p(X|\Theta) - \log p^*(X|\Theta)\):
Is the likelihood along the best path a good approximation of the real likelihood of a sequence given a model?

11 Mixtures of Gaussians and HMMs

HMMs with Gaussian mixture emission pdfs should be trained and used for recognition of utterances of English digits from ‘one’ to ‘five’. Go through the tutorial “Mixtures of Gaussians” and download the accompanying MATLAB programs and data.

Add to the end of the report the MATLAB script you programmed for producing all the results. Please provide the Name and Matrikelnummer.

1. Load the signals into MATLAB using load digits, and play them with the MATLAB functions sound or wavplay, e.g., sound(three15).

Process the speech signals to get parameters (features) suitable for speech recognition purposes. For the signals loaded from digits.mat this is done using the function preproc() (without arguments). This function produces a cell array data{N} for each digit N holding the parameter vectors (mel-frequency cepstral coefficients, 12-dimensional) for each training signal (e.g., data{1}{2} holds the sequence of parameter vectors for the second example of digit ‘one’), as well as a cell array testdata{N} for each digit N for each test signal. preproc() uses functions that are part of VOICEBOX, a MATLAB toolbox for speech processing.

The sequences of parameter vectors have different length (as also the speech signals differ in length!), that is why we can not store all sequences for training or testing in one array.

2. Train one HMM for each digit. Training of HMM parameters (emission pdfs using the EM algorithm, as well as prior and transition probabilities) is done using the function train:

   >> [HMM] = train(data,K,Ns,nr_iter)

where Ns denotes the number of HMM states, K the number of Gaussian mixtures in the emission pdfs, and nr_iter the number of iterations. The function trains left-to-right HMMs with covariance matrices of the Gaussian mixtures in diagonal form.

3. Determine the recognition rate on the test signals. To test the model use the function recognize as described in the tutorial, e.g., for digit 1:

   [r1,L1,p1] = recognize(testdata{1},HMM_1,HMM_2,HMM_3,HMM_4,HMM_5)

4. In your report note down your chosen settings, intermediate results and considerations, and the recognition results (recognition rate for each digit, and for the whole set). Which digits seem more easy to recognize? Which digits get easily confused during recognition? Use different values for the number of states (Ns=2...5) and the number of Gaussian mixture components (K=1...3). How do these numbers affect the computational effort for training?
and the recognition rate? Which values for $N_s$ and $K$ do you think are optimal? Please present your recognition results (for each digit and for the whole data set) as well as the time for training the models as tables/figures (depending on the choice of $N_s$ and $K$). For measuring the training time use the matlab commands `tic` and `toc`.

5. Why does it seem reasonable to use left-to-right HMMs for this task, and for speech in general? What are the advantages/disadvantages? Please modify the function `train` to train ergodic models and do experiments similar as in task 4 for $N_s=2\ldots.5$ and $K=1\ldots.3$. Again, present the tables for the recognition rate and the computational effort for training.

6. Why do we use diagonal covariance matrices for the Gaussian mixtures? What assumption do we take, if we do so? Please modify the function `train` to train left-to-right models with a full covariance matrix. Again, do experiments similar as in task 4 for $N_s=2\ldots.5$ and $K=1\ldots.3$ and present the tables for the recognition rate and the computational costs for training. Give also a table for the number of parameters we have to train in this case compared to the left-to-right HMM using diagonal covariance matrices for the Gaussian mixtures (as in task 4). Is there a connection between the number of parameters we have to train, the number of training samples we have available, and the recognition performance? Please present a short discussion on that in your report.

7. What kind of model do we have if we have just $N_s=1$ state in our HMM. Please give the equation $P(X|\Theta)$ for this model in terms of $\Theta = \{\pi, A, B\}$.

12 Adaptive Filters

Your homework should contain a printout of your MATLAB function `lms2`, informative plots (in particular the plots requested in the text below) in an appropriate scale and with the axes labeled, along with your observations written down (whole sentences). Add to the end of the report the MATLAB scripts you programmed for producing all the results.

Please provide the Name and Matrikelnummer in the scripts and in the body of the email.

1. Write a MATLAB function `[y,e,c]=lms2(x,d,N,mu)` which implements an adaptive transversal filter using the LMS adaptation algorithm (see Tutorial). Start with the following header:

```matlab
function [y,e,c] = lms2(x,d,N,mu)
% [y,e,c] = lms2(x,d,N,mu)
% Adaptive transversal filter using LMS (for algorithm analysis)
% INPUT
% x ... vector containing the samples of the input signal x[n]
% size(x) = [xlen,1] ... column vector
% d ... vector containing the samples of the desired output signal d[n]
% size(d) = [xlen,1] ... column vector
% N ... number of coefficients
% mu .. step-size parameter
% OUTPUT
% y ... vector containing the samples of the output signal y[n]
```

[mailto:paul.meissner@student.tugraz.at](mailto:paul.meissner@student.tugraz.at)
2. For a system identification application, write a MATLAB script to visualize the adaptation process in the time domain. Use your function `lms2()` and let \( x[n] \) be normally distributed random numbers with zero mean value and variance one (use the MATLAB function `randn()`). For the unknown FIR filter, you should add to each digit of your Matrikelnummer \( 0.5 \) and use this as impulse response \( h \). Choose an appropriate filter length \( M \). To calculate \( d[n] \), use the MATLAB function \( d = \text{filter}(h,1,x) \). Take \( N = M \) and choose a proper value for the step-size parameter \( \mu \).

Compare the output of the adaptive filter to the desired signal in one plot, compute the error signal, and plot the squared error signal (learning curve) using a logarithmic scale (e.g., using `semilogy`, or by converting the value of the squared error to dB). Be sure to use long enough signals to arrive at the minimum of the learning curve.
3. Modify your script and examine the cases \( N > M \) and \( N < M \). Plot the coefficients \( h \) of your reference filter, along with the coefficients found by the LMS algorithm after several hundred sample times, e.g.:

\[
\begin{align*}
\text{>> } x &= \text{randn}(1000,1); \\
\text{>> } h &= \text{digits(your Matrikelnummer)}; \\
\text{>> } c &= \text{rls1(x,h);}
\end{align*}
\]

Do the coefficients converge to fixed values for the case \( N < M \)?

4. Add a random signal to the desired output \( d[n] \) (this would be the signal from a local speaker in an echo cancellation application):

\[
\text{>> } d[n] &= d[n] + 1e^{-6} \text{randn(length(d),1)} \\
\]

Take again \( N = M \), and plot the learning curve for the LMS algorithm using the noisy desired output signal. Compare to the learning curve found before (without noise added). Do the coefficients converge? Use different step-size parameters \( \mu \) (in the range of \( 0 < \mu < \frac{2}{\|x[n]\|^2} \)) and plot the learning curves. Describe your observations concerning the convergence speed and the excess error. What is a good value for \( \mu \).

5. Use the function \texttt{rls1()} implementing the recursive least squares adaptation algorithm (this function is provided) instead of \texttt{lms2()}, choose \( \rho \) appropriately, and compare the two algorithms concerning the time until the coefficients converge (plot the two learning curves).

6. For the two-dimensional case \( N = M = 2 \), plot the adaptation path of the coefficients in the \( c \)-plane: \( c[n] = (c_0[n], c_1[n])^T \).

Take the last two digits of your Matrikelnummer and add to each digit 0.5. Take this as impulse response \( h = (h_0, h_1)^T \), with \( h_0, h_1 \neq 0 \).

Use both algorithms (\texttt{lms2()} and \texttt{rls1()}) and the following input signals:

\[
\begin{align*}
&\text{(a) } x[n] = \text{sign}(\text{rand}[n] - 0.5) \quad \text{and} \quad \mu = 0.5, \rho = 0.95 \\
&\text{(b) } x[n] = \text{randn}[n] \quad \text{and} \quad \mu = 0.5, \rho = 0.95 \\
&\text{(c) } x[n] = \cos(\pi n) \quad \text{and} \quad \mu = 0.5, \rho = 0.95 \\
&\text{(d) } x[n] = \cos(\pi n) + 2 \quad \text{and} \quad \mu = 0.1, \rho = 0.95
\end{align*}
\]

Do not forget to re-compute your reference signal \( d[n] \) for each input signal. Compare the results for the four input signals. Describe your observations.