Machine Learning for Signal Processing
Homework 3 - Part B

Bhiksha Raj - Abelino Jimenez, Anurag Kumar, Yixuan Zhang, Davy Uwizera

This is Part B of homework 3. This part contains 2 questions.

1 Predicting the Election

In this problem we will try to track a number of opinion polls and try to estimate the *true* support for the candidates in a recent election.

The election is between four candidates. Public sentiment about the candidates fluctuates all the time. A number of opinion polls try to gauge public sentiment. However, since opinion polls are fundamentally noisy procedures (affected by factors such as the specific subset of people they poll, or the number of samples in their poll), each of them can be viewed as a noisy measurement of the true public sentiment. We will try to obtain a better estimate of the true sentiment, as well as the uncertainty of the estimate (which a pollster could use to establish a margin of error).

We will model the polls as the output of a linear Gaussian process as follows:

- Let S_t represent the *state* of public sentiment. In our case, its a 3-dimensional vector representing the percent of public that currently favors candidates 1, 2 and 3. Although there are 4 candidates, we need not model the 4th explicitly, since the fourth is linearly dependent on the first three – the four must sum to 100.
- We expect public opinion to generally stay consistent in the absence of other effects. So our *state equation* is given by
$$S_t = S_{t-1} + \epsilon_t$$
where ϵ_t is the *innovation* in the time period t that changes the state.
  1. We will assume that *a priori* probability distribution of S (i.e. before any opinion polls) is a Gaussian with mean $\bar{S}_0$ and variance R_0: $P(S_0) = \mathcal{N}(S_0; \bar{S}_0, R_0)$.
  2. We will assume that the innovation ϵ is also Gaussian with mean 0 and variance Θ_ϵ, i.e. $P(\epsilon) = \mathcal{N}(\epsilon; 0, \Theta_\epsilon)$.
- Let O_t represent the vector of opinion poll measurements. We will consider 17 different polls. So, in our problem O_t is a 51-dimensional vector (since each poll reports numbers for 3 candidates). *However not all polls are obtained each week. So for some weeks, some polls may be absent, in which case the dimensionality of the observation will be $3K$, where K is the number of "obtained" polls and will be less than 17.*
- We model the opinion polls as a noisy measurement of S_t given by
$$O_t = AS_t + \gamma_t$$
where A is a $D \times 3$ "observation matrix", and D here is the number of collected opinions at time t (which, for K polls, will be $3K \leq 51$). γ is an observation noise, and assumed to have a Gaussian distribution with mean μ_γ and variance Θ_γ, i.e. $P(\gamma) = \mathcal{N}(\gamma; \mu_\gamma, \Theta_\gamma)$.

Our objective is to use the measurements $O_{0:t}$ (i.e. all measurements from time 0 to t) to estimate the true sentiment S_t at time t.

Problem 1.1

Write out the Kalman filtering equations to estimate S_t at each time t.

Problem 1.2

Implement the Kalman filter (you must submit the code). Run it on the provided data series (which only comprises the sequence of observations $O_0, \cdots, O_T$) and predict the true S_t at each t. Plot the estimated S_t as a function of time (this will be a single plot with 3 curves). Submit both the plot, and the the estimated state at every time. Also submit the final state uncertainty (i.e. the variance matrix of the state).

Problem 1.3

You wont be scored on this, but compare the final estimate (at the final instant) with the true voting percentages in the 2016 presidential election. You can get this from `http://www.realclearpolitics.com/elections/live_results/2016_general/president/`the RealClearPolitics webpage on 25th November or later for the final count (or close to it).

Data for the problem

The data can be found in the directory `hw3bmaterials/problem1` and includes the following:

- A file called `opinionpoll.mat`. Each row of this file is the set of opinion poll numbers for one week. Some numbers are `NaN`. These represent opinion poll numbers that were *not* obtained, i.e. in that particular week the actual number of opinion polls obtained was less than 17, and so the `NaN` entries represent polls that were not taken. Note that since each poll gives you three numbers (one per candidate), the `NaN`s will occur in groups of three. You will have to keep track of which data entries are missing in any week, because you will have to remove that entry from the observation to get the true observation vector, and also remove the corresponding row from A, and the corresponding columns and rows of Θ_γ to get the actual covariance matrix for the observation noise that week.
- A file called `prior.mat` with the parameters of the *a priori* probability distribution of S_0. The first line in this matrix contains $\bar{S}_0$. The second line contains the *diagonal* entries of R_0. The off-diagonal entries are assumed to be 0.
- A file called `epsilon.mat` with the *diagonal* entries of Θ_ϵ. The off-diagonal entries of Θ_ϵ are assumed to be 0.
- A file called `gamma.mat`. The first row of the file gives you μ_γ. The second row has the *diagonal* entries of Θ_γ. The off-diagonal entries of Θ_γ are assumed to be 0.

N.B: Please remember that this is only a homework problem and may not in any way be indicative of reality. Our model is unrealistic – its unlikely that either the noise nor the innovation is Gaussian. We're also not explicitly handling other factors that affect the polling, or the constraint that the samples are strictly non-negative (you can't have a negative percent of the population voting for anyone). Various other factors are being ignored (although, in principle, all of these could be included in the model). Nonetheless, we believe the computational exercise itself is interesting and should tell you something of the power of MLSP techniques.

Submission Instructions

1. Kalman Filter Equations should be in your report
2. Write one main script `p1Main.m`, which should do everything when we run it. If you implement any function scripts which is called by this main script, submit those functions as well.
3. Submit the plot and result matrices in `results` folder of problem 1. Name your matrices properly so that we can identify which one is what.

2 Linear Discriminant Analysis

In this problem we will explore the use of *Linear Discriminant Analysis* for the problem of language identification.

The goal is to build a language recognition system based on "*I-vector*" representation of speech signals. I-vectors are a factor-analysis based mechanism for representing audio recordings as fixed-length vectors. They are particularly useful for categorizing sounds, specifically speech, but also other data, by their categorical content (e.g. gender, language, speaker ID).

In the folder `hw3bmaterial/problem2/data` you will find three directories: `Train`, `Dev` and `Eval`. These directories correspond respectively to the training data that will be used to train your classifier, development data to tune your classifier and evaluation data to evaluate the different models. In each directory you will find 24 files that correspond to 24 different language classes. Each line of these 24 files corresponds to a single I-vector of dimension 600 and represents a single speech recording. The goal of this problem is to build a classifier, which is based on Linear Discriminant Analysis (LDA).

Problem 2A: Linear Discriminant Analysis

Implement LDA (linear discriminant analysis) and cosine scoring for the I-vectors in the data set. The classification algorithm based on LDA and cosine scoring is described as follows:

- Normalize the length of each I-vector as:

$$\mathbf{w} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$$

  We will work only with the normalized I-vectors.

- **Training LDA**:
  - Compute the global mean of all the (normalized) vectors:

$$\bar{\mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{w}_i$$

    where n represents the total number of vectors across all languages (classes).
  - For each language l compute the class mean of all the I-vectors in the class:

$$\mathbf{w}_l = \frac{1}{n_l}\sum_{\mathbf{w}:\mathbf{w}\in l}\mathbf{w}$$

    Here n_l is the number of vectors in language l, and $\{\mathbf{w} : \mathbf{w} \in l\}$ represents the set of all vectors from language l.
  - Compute a *between class* covariance matrix:

$$\mathbf{S}_b = \sum_{l=1}^{L} n_l(\mathbf{w}_l - \bar{\mathbf{w}})(\mathbf{w}_l - \bar{\mathbf{w}})^\top$$

    where L is the total number of languages (classes).
  - Compute a *within class* covariance matrix:

$$\mathbf{S}_w = \sum_{l=1}^{L}\sum_{\mathbf{w}:\mathbf{w}\in l}(\mathbf{w} - \mathbf{w}_l)(\mathbf{w} - \mathbf{w}_l)^\top$$

    Here, the outer sum is over languages, and the inner sum is over all vectors from that language.

The goal of LDA is to estimate an LDA matrix $\mathbf{V}$ that solves the following *generalized* Eigenvalue problem:

$$\mathbf{S}_b\mathbf{v} = \lambda\mathbf{S}_w\mathbf{v}$$

You can solve the above generalized Eigenvalue problem using any Eigen value solver. In matlab you can use the command

```
[V,D] = eigs(Sb, Sw, L-1)
```

If you have L different classes, LDA will provide $L-1$ non-zero Eigenvalues. The $\mathbf{V}$ matrix we want to compute is obtained from the $L-1$ Eigenvectors corresponding to the $L-1$ non-zero Eigenvalues: this will give us a $L-1 \times D$ matrix $\mathbf{V}$ (where D is the dimensionality of the data, in our case the I-vectors). Note that the columns of $\mathbf{V}$ are orthogonal to one another.

Training the classifier

1. Project the (normalized) I-vectors onto the columns of $\mathbf{V}$ and the renomalize the length:

$$\hat{\mathbf{w}} = \frac{\mathbf{V}^\top\mathbf{w}}{\|\mathbf{V}^\top\mathbf{w}\|}$$

2. For each class l compute the mean and its normalized length.

$$\mathbf{m}_l^{raw} = \frac{1}{n_l}\sum_{\hat{\mathbf{w}}\in l}\hat{\mathbf{w}}\mathbf{m}_l = \frac{\mathbf{m}_l^{raw}}{\|\mathbf{m}_l^{raw}\|}$$

   $\mathbf{m}_l$ is the "model" for language l.

Testing the classifier

1. Project the test I-vectors onto the columns of $\mathbf{V}$ and the renomalize the length:

$$\hat{\mathbf{w}}_{test} = \frac{\mathbf{V}^\top\mathbf{w}_{test}}{\|\mathbf{V}^\top\mathbf{w}_{test}\|}$$

2. Classify the test vector as belonging to the language (class) whose mean is closest to it in angle, i.e. assign the test vector to the language whose model mean has the greatest inner product with it:

$$\hat{l}(\hat{\mathbf{w}}) = \arg\max_l \left(\mathbf{m}_l^\top\hat{\mathbf{w}}_{test}\right)$$

Homework Problem

1. Train an LDA projection matrix $\mathbf{V}$. You will have to submit $\mathbf{V}$.

2. Train models for all the classes.

3. Classify each of the test recordings in both the "dev" and "eval" directories. Return the classification output.

4. Report dev accuracy:

$$Acc_{dev} = \frac{\text{number of correctly classified vectors in dev directory}}{\text{total number of vectors in dev directory}}$$

5. Predict the output for all points in the Evaluation set. Eval.txt contains 2400 test instances. Predict the output for each and submit it in the results folder.

6. Your grade will depend on your accuracy on the Eval set.

Submission Instruction

1. Write a function `trainLDAclass`. This function should return the "model" ($\mathbf{m}_l$) for each language l. You are free to decide the inputs to this function

2. Write a main training script `trainLDAMain`. This script when run should train the model for each class.

3. Write a script `testDev` which does the testing on Dev set and produces the accuracy on the Dev set. Report this accuracy in your report.

4. Write a scrip `testEval` which does testing on Eval set. Save the results in a .mat file, `evalResults.mat` and submit this result file. It should be a vector of 2400×1 dimensions, that is the class prediction for each class. Each row should be number between 1 to 24 representing the predicted class. **DO NOT** change the order of evaluation instances. We will simply load your results matrix and compute accuracy using the ground truth matrix. Your grade depends on the accuracy on the Eval set. The `evalResults.mat` file should be inside the `results` folder of problem 2.

5. Submit all scripts and the `evalResults.mat` file.