Checkpoint 12.1

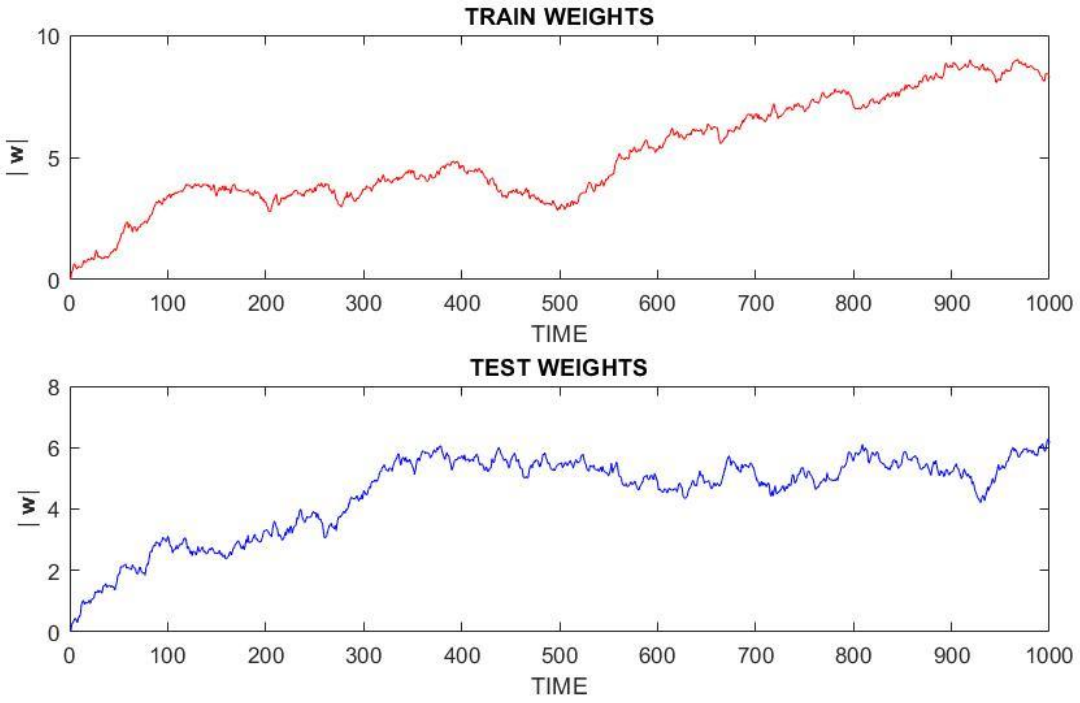
Use the matlab script main12a.m to create a sequence of weights for a dynamic linear model, using Eq. (8) with a specified 'teacher' α_0. Inspect the sequence of weights, why is the magnitude increasing? Generate a similar test sequence also based on α_0.

Based on the two sequences of weights we generate sequences of training and test observations $(t_n, \mathbf{x}_n)$. For the observations, we simulate i.i.d. normal input and additive noise with precision β. Using the dynamic updates we will make predictions on train and test sets for specific choices of α and β. Explain how we can use training set predictions to estimate the optimal combination of (α, β).

In order to respond the question about why the weights increase over time we should take a look first at the following equation that shows a Markovian random walk that produces its values:

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \boldsymbol{\nu}_n \qquad \text{with } \boldsymbol{\nu}_n \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$$

At first glance one could think that, as the mean is equal to 0, this random walk should converge towards 0, but the results show otherwise. The answer to this enigma resides in the additive nature of a random walk. The previous formula can be formulated as an addition of gaussian distributions overtime. Thus, for example, let's run a simple experiment where n = 2. In this case $w_2 = 0 + A + B$ being A and B normal distributions defined previously. If we perform this addition we obtain $w_2 = \mathrm{N}(\mu_A + \mu_B, {\sigma_A}^2 + {\sigma_B}^2)$ and as it is given that mean is equal to 0 and the standart deviations are the same for both we got that $w_2 = \mathrm{N}(0, 2\sigma^2)$ and in the n case $w_n = \mathrm{N}(0, n\sigma^2)$. Thus, the root mean square error (and standart deviation in this case) becomes $\mp\sigma\sqrt{n}$, increasing this way with the number of sumples. This can be visualized below:

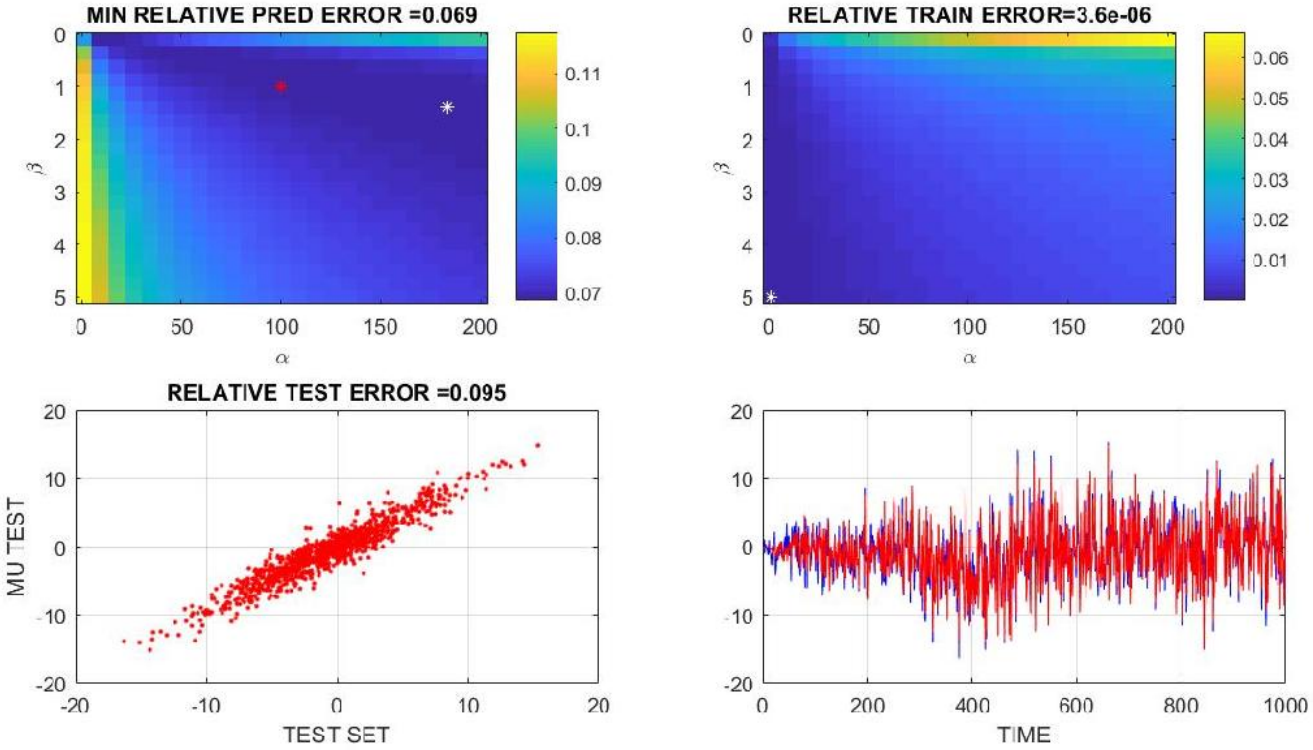


Once the weights of the test and train data are computed, we can also compute the target values for both cases following equation (1) $t = \mathbf{w}^{\top}\mathbf{x} + \epsilon$.

Once we have our datasets in place, we can use the dynamic updates to find the optimal values of beta and alpha. For this matter, we will create two array with different values for each and iterate through the values. In each of this iteration we will go through the process of predicting the target value for a certain time n using $\mu_{t_{N+1}} = \mu_p^{\top}\mathbf{x}_{N+1}$, and then we will update the weights using the un-normlized posterior

$$\begin{aligned}\boldsymbol{\mu}_{\mathbf{w},n} &= \left((\boldsymbol{\Sigma}_{\mathbf{w},n-1} + \alpha^{-1}\mathbf{I})^{-1} + \beta\mathbf{x}_n\mathbf{x}_n^{\top}\right)^{-1}\left((\boldsymbol{\Sigma}_{\mathbf{w},n-1} + \alpha^{-1}\mathbf{I})^{-1}\boldsymbol{\mu}_{\mathbf{w},n-1} + \beta t_n\mathbf{x}_n\right)\\ \boldsymbol{\Sigma}_{\mathbf{w},n} &= \left((\boldsymbol{\Sigma}_{\mathbf{w},n-1} + \alpha^{-1}\mathbf{I})^{-1} + \beta\mathbf{x}_n\mathbf{x}_n^{\top}\right)^{-1}\end{aligned} \tag{14}$$

. Now, for each of the values of beta and alpha we will compute both the estimation and prediction errors, using the minimum of this last one to select the optimal value of the parameters. This values will be applied afterwards to the test set in order to check the test error of the model. All the results can be visualized in the following graphs:



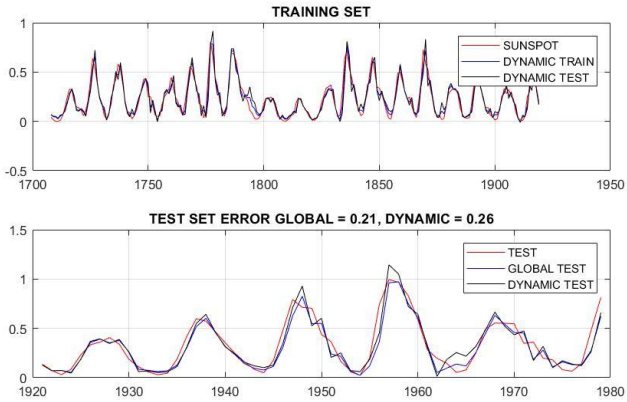
In the first two graphs we have plotted both the prediction and the train error for all the set of beta and alpha values, respectively. We have also plotted the optimal pair of hyperparameters based on each metric, as well as the true values of alpha and beta in red. Out of this results we can see that the train error is not a useful metric to predict the optimal values of alpha and beta. The reason behind this is because the relative train error is computed using the mu estimated for x_1:n and the train error is computed with x_1:n-1. This assumption holds in the static case, when w_n=w_n-1, but in this case we have the random walk with variance alpha. Actually, it explains why that the optimal training error alpha is really close to 0.

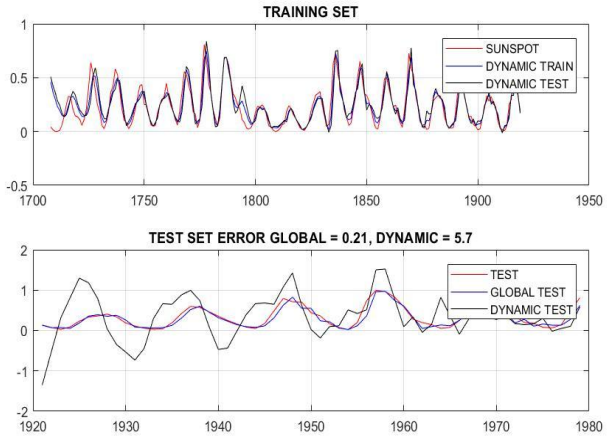
From the graphs at bottom, we can see how the target test values are really close to the predicted test values. We can also see this relation presented in the shape of a signal in the plot on the right, where we can see how the predicted values are similar to the target ones.

Checkpoint 12.2

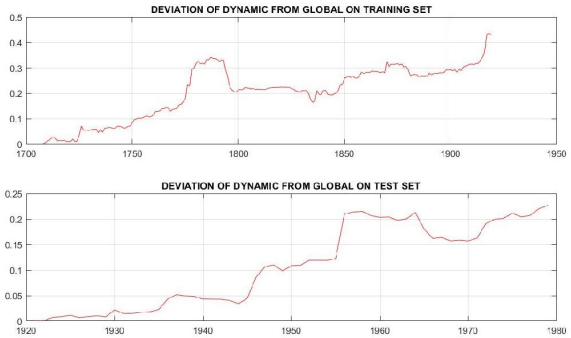
Use the matlab script main12b.m to investigate the dynamic linear model on the sun spot data for fixed (α, β). Inspect the deviations of the dynamic linear model from the global linear model. Is the random walk prior, i.e., a non-stationary model with 'growing weights', useful in the sunspot case?

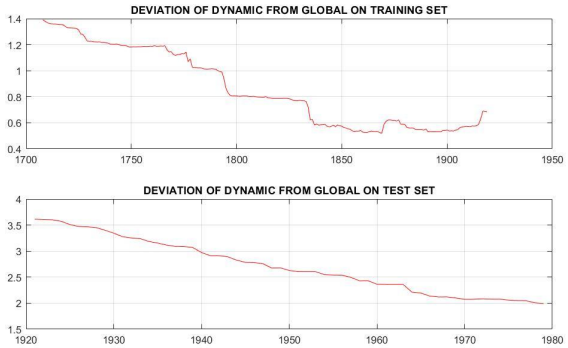
In this exercise we are going to compare both the global static prediction and the dynamic predictions. In order to do so, first we are going to solve the linear system using the strategy out of exercise 3. The results out of this method will be used not only as a stand alone solution, but to give a "hot start" to the weights of our dynamical model. The improvements of using this method instead of a cold start are significant, as we can note from the results below





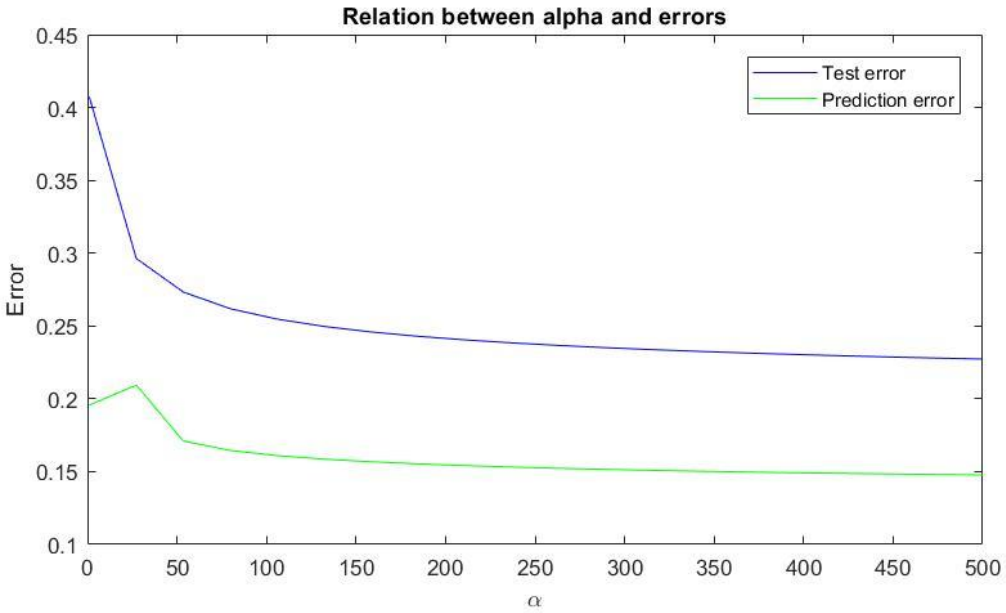
We can also note that this is the case when we look at the deviation of the weights from the global model.





Given this results, we have to ask ourselfs if this dynamic model is the best for this dataset, as the results that we saw (even with the hot start) are worse than the ones out of the global model. When we introduce the random walk prior into the model our weights start to deviate from the the global weights, and we start to see that the test set error is directly correlated with the distance to the of the global model weights.

In order to probe this, we are going to study the relation of the alpha parameter (the one that goberns the random walk of the weights) with the errors:



We can see that when we increase the value of alpha (i.e. the difference in the weights is smaller and the system behaves ore like the global system) both training and prediction error decreases.

Checkpoint 12.3

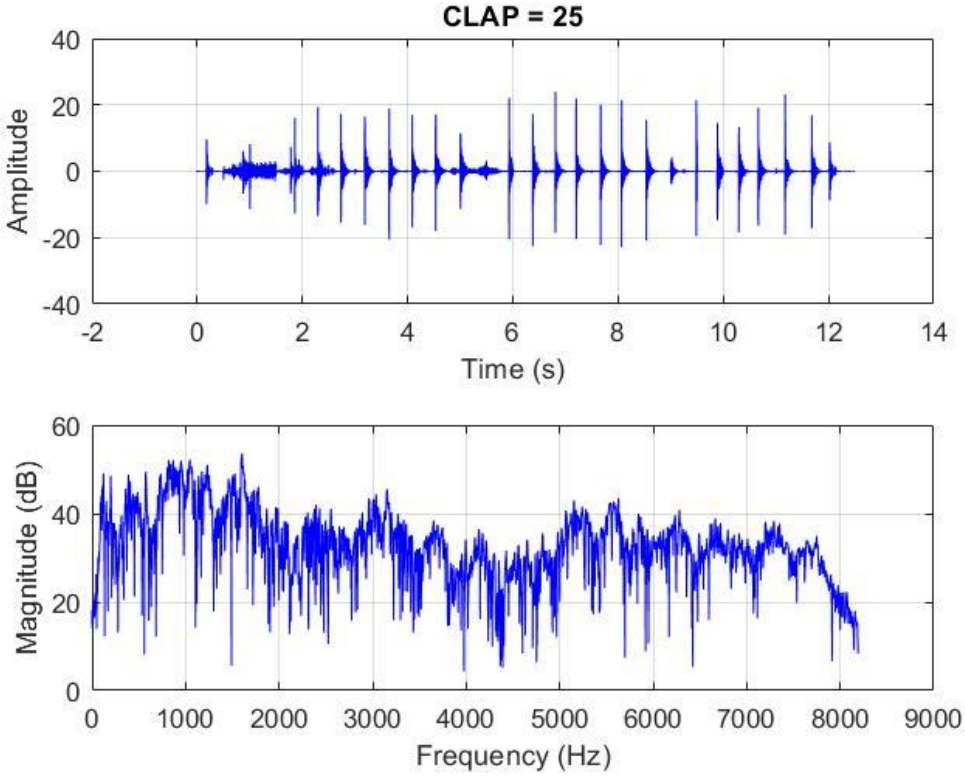
We finally use a linear model to classify sounds in a realtime setting. Use the matla script main12c.m to acquire sound in blocks of length T. We will use linear model t classify sounds in two classes labeled by targets $t = +/-1$.

The basic features are derived from the frequency content (using a window Fourie transform and principal component based dimensional reduction).

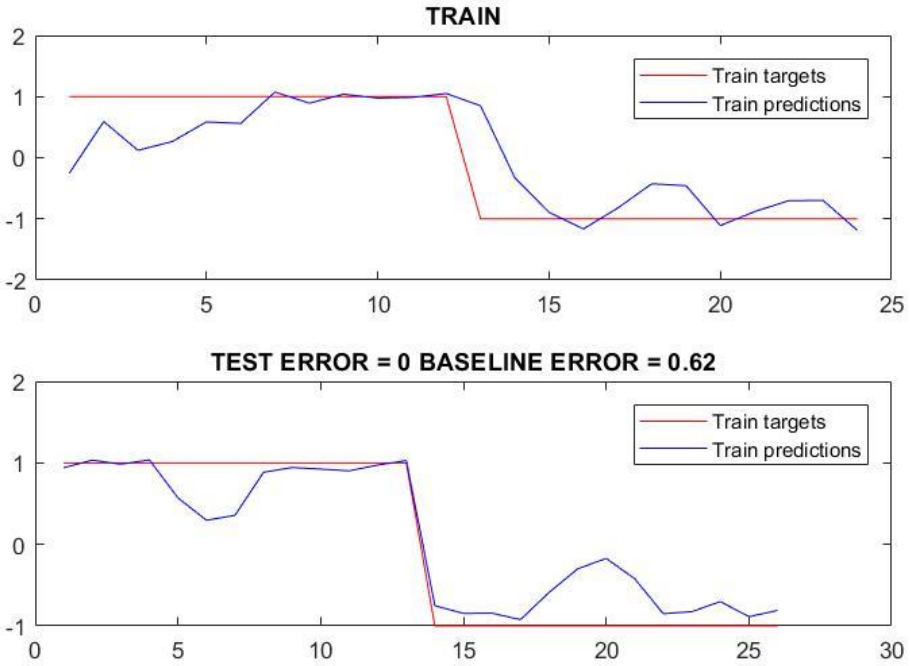
We first pre-train the classifier by creating two sound sequences, e.g., an 'impact soun from tapping the table with a pen and a clapping sound.

Following the training, start executing new sounds for 60 seconds and inspect th resulting classifications in the Matlab window.

In this exercise we will apply a linear model to classify sounds recorded in a window setting. In order to do so we are going to record two different sequences of audio, one from a clap and one from an impact. For each of this windows we will apply a fft and we will save its value in our dataset. Finally, we will apply a principal component decomposition to each sample and we will end up with the 4 main dimensions of the data.



Once our dataset is ready we will split it into training and test, and we will run our linear classifier over the training data, with labels of 1 for clap and -1 for impact. This will give us a set of weights that will allow us to classify in real time between the two classes.



From this results we can see how the model is able to predict all the test samples correctly, with quite low uncertainty. However, it is important to realize that the quality of this model is highly influenced by

the data collected, as it is such a small dataset. Thus, if we miss to make a sound in both classes it may affect the performance of the model greatly.