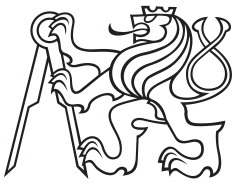
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of cybernetics

Probabilistic model for land cover type recognition in temporal sequences of satellite images

Martin Španěl

Supervisor: doc. Boris Flach
May 2019

Acknowledgements

I would like to thank my supervisor Boris Flach for his great support, good advice and deep insight into HMM and machine learning in general. I am also grateful to Lukáš Brodský for sharing his knowledge of satellite imagery and valuable feedback.

I very much appreciate all of the teachers that taught me at CTU and CU, because without them I would not be able to solve most of the problems in this thesis.

Thanks should also go to my family and friends who always supported and encouraged me.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 24, 2019

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 24. května 2019

Abstract

We developed a method to recognize changes of the land cover in sequences of satellite images with high temporal resolution. Two machine learning approaches are proposed: one based on a combination of several hidden Markov models, the other based on a single bidirectional recurrent neural network. Both methods were tested on real data from the Landsat satellites.

Keywords: povrch krajiny, satelit, HMM, RNN, body změny, Landsat, Sentinel

Supervisor: doc. Boris Flach

Abstrakt

Vyvinuli jsme metodu na rozpoznávání změn povrchu krajiny v posloupnostech satelitních snímků s vysokým časovým rozlišením. Představujeme dva postupy strojového učení: Jeden založený na skrytých Markovových modelech, druhý založený na obousměrné rekurentní neuronové síti. Obě metody byly otestovány na skutečných datech ze satelitů Landsat.

Klíčová slova: land cover, satellite, HMM, RNN, breakpoints, Landsat, Sentinel

Překlad názvu: Pravděpodobnostní model pro rozpoznání druhu využití krajiny v časových sekvencích satelitních snímků

Contents

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Španěl Martin** Personal ID number: **470084**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer Vision and Image Processing**

II. Master's thesis details

Master's thesis title in English:

Probabilistic Model for Land Cover Type Recognition in Temporal Sequences of Satellite Images

Master's thesis title in Czech:

Pravděpodobnostní model pro rozpoznání druhu využití krajiny v časových sekvencích satelitních snímků

Guidelines:

1. Design a probabilistic model that relates temporal sequences of satellite measurements with land cover types. Conceptually, such a model can be for instance based on a combination of Hidden Markov Models and Variational Autoencoder Networks.
2. Derive the corresponding inference and learning algorithms.
3. Implement the algorithms (in Python/Pytorch) and test the approach on the data obtained from Sentinel satellites for a selected region in CZ (after pre-processing). The annotations requiered for parameter learning will be obtained from a previously develeped baseline method, subsequently validated by an expert.
4. Validate the resulting approach and compare it with the baseline method.

Bibliography / sources:

[1] Jan Verbesselt et al., Detecting trend and seasonal changes in satellite image time series, Remote Sensing of Environment 114 (2010).
[2] Carl Doersch, Tutorial on Variational Autoencoders, arXiv:1606.05908, 2016
[3] P. Benedetti et al., M3Fusion: A Deep Learning Architecture for Multi-{Scale/Modal/Temporal} satellite data fusion, arXiv:1803.01945 (2018).
[4] K. Cho, B et al., Learning phrase representations using RNN encoder-decoder for statistical machine translation, proceedings of EMNLP 2014, pp. 1724–1734.

Name and workplace of master's thesis supervisor:

Boris Flach, Dr. rer. nat. habil., Machine Learning, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **09.01.2019** Deadline for master's thesis submission: **24.05.2019**

Assignment valid until: **30.09.2020**

Boris Flach, Dr. rer. nat. habil.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Chapter 1
Introduction

The motivation for the proposed methods is to find changes in the land cover types based on data obtained from Sentinel-2 satellites [4]. The satellites provide optical measurements of the whole planet with both high temporal resolution (approximately one measurement per five days for any location on earth) and spatial resolution (from 10 meters per pixel) in 13 spectral bands.

Not every measurement is useful, because in some of them the land is occluded by clouds. The clouds need to be masked with a different algorithm that is not in the scope of this work. Consequently, the algorithms for breakpoint detection and land cover type classification need to be able to handle data with missing entries.

The input of the proposed algorithm is a set of temporal sequences of measurements for individual locations. If we interpret the sequence of satellite images as video, each of these sequences contains the sequence of changing colors on individual coordinates of the frame.

The output is a classification for every combination of time and location. The classification can be also interpreted as a segmentation of the individual temporal sequences and the classification of the individual segments.

The classification should be consistent in time for any location, unless sudden changes of the land cover type occur in the location. In such case, the time of the change needs to be precisely recognized. The recognizer should not get confused by changes of land cover appearance caused by seasonal changes and it needs to handle input sequences with missing entries.

1.1 Challenges in the task

The data obtained from Landsat or Sentinel-2 satellites have interesting properties. The very high temporal resolution allows to recognize changes in the past with high precission. However the spatial resolution is much lower than it would be for example in aerial images. For this reason, recognition with precission to individual pixels is desired. Recognizing the spatial boundary of a land cover type with an error of one pixel is an error of ten meters in real world, which is significant.

For this reason, we did not classify patches of the images, but rather individual pixels. To reduce the risk of misclassification on the boundaries,

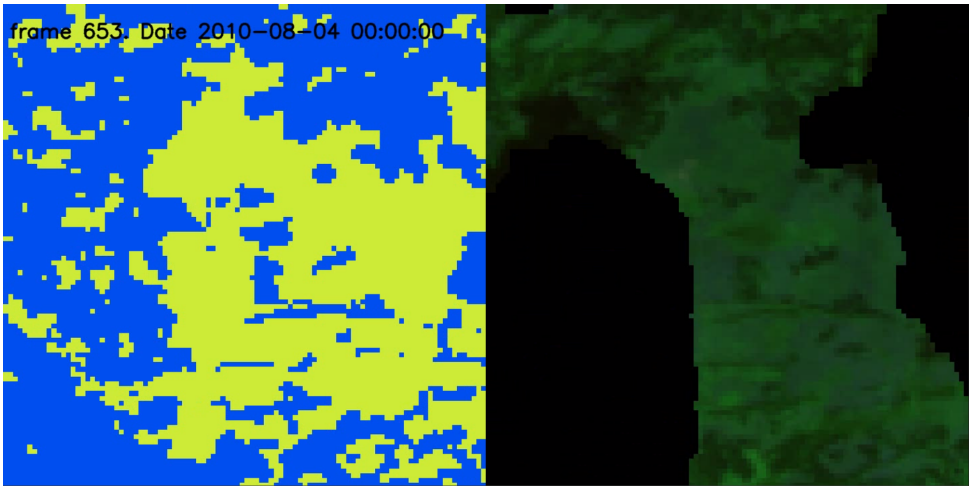


Figure 1.1: Example of a classification. On the right, there is one of the sequence of satellite images. Some of the pixel values were removed, because the land was covered by clouds in them. On the left, there is a classification of the pixels - the output of a classifier.

we recognized the classes in individual locations, which each had a single corresponding pixel in every satellite image, separately. So all of the proposed algorithms do not take a sequence of images as input, but rather sequences of individual pixel values corresponding to some location changing in time.

Although we classify the sequences for individual pixels, it is trivial to compile the classifications to classifications of images for the dates, when the satellite images were taken and visually compare them. Figure 1.1 shows a possible classifier output and input.

Two kinds of tasks were considered. First, we propose breakpoint detection agnostic to the classes by an unsupervised algorithm. The detected temporal segments can then be classified with another algorithm.

Second, we propose a breakpoint detection algorithm that uses models for the individual classes to reconize breakpoints and classify the segments at the same time.

Chapter 2

Breakpoint detection

2.1 Problem definition

The core problem that we are solving in this work is detecting breakpoints in a sequence. Consider a sequence of vectors $x_1, \ldots, x_n$. Breakpoints are a growing sequence of k indices $T = \tau_1, \ldots, \tau_k$ from the interval 1 to n. The sequence of breakpoints may be empty.

To the sequence of breakpoints corresponds unambiguously a segmentation $S(T) = \{(1, \tau_1), (\tau_1, \tau_2), \ldots, (\tau_k, n)\} = \{s_1, s_2, \ldots, s_{k+1}\}$. Every s_i is an interval and we will call it a temporal segment and it defines a subsequence of x, we will denote it x_{s_i}.

In breakpoint detection, we seek breakpoints for a given sequence of vectors x that maximize some probability $P_B(x, T)$. In general, the loss function should enforce that in every segment s_i, x_{s_i} fits some model (possibly with different parameters for every segment, while keeping the number of breakpoints k as low as possible.

Let us suppose that we have a probabilistic model that can evaluate the probability $P_S(x)$ of any subsequence of x in that model, assuming that there are no breakpoints in it. We will call such model a segment model.

We can use the segment model to define a more complex probabilistic model describing sequences with breakpoints. We will call it a breakpoint model. It will be defined as follows:

$$P_B(x, T) = p^{-|T|} \prod_{s \in S} P_S(x_s), \tag{2.1}$$

where p is a constant parameter called a breakpoint penalty. We will show an algorithm that maximizes likelihood with efficiently.

2.2 loss optimization algorithm

The breakpoints minimizing equation (2.1) can be found also by maximizing the negative logarithm of the loss, that is

$$\max_{T} \left(-\log(p)|T| + \sum_{s \in S(T)} \log(P_S(x_s)) \right). \tag{2.2}$$

Notice that we can split the minimization to two steps:

$$\begin{aligned} \max_{k} \left(\max_{T:|T|=k} \left(-\log(p)|T| + \sum_{s \in S(T)} \log(P_S(x_s) \right) \right) = \\ \max_{k} \left(-k\log(p) + \max_{T:|T|=k} \sum_{s \in S(T)} \log(P(x_s)) \right). \end{aligned} \tag{2.3}$$

Let us define auxiliary function that will represent the log-probability of the most likely model on a segment. $d(m, i, j)$ will be the log-probabilities on the interval (t_i, t_j) if there are m breakpoints allowed in the interval, with no breakpoint penalty.

The function can be defined recursively:

$$d(0, i, j) = \log(P_S((x_i, \ldots, x_j))), \tag{2.4}$$

$$d(m, i, j) = \max_{b \in (i,j)} d(m-1, i, b) + d(0, b, j), \tag{2.5}$$

It is easy to see that

$$d(k, 0, n) = \max_{T:|T|=k} \sum_{s \in S(T)} \log(P(x_s))). \tag{2.6}$$

We can compute the values d(k, 0, n) for all possible values of k efficiently using dynamic optimization as described in [2]. If some upper bound for the number of breakpoints is assumed, we can examine only those values of k.

First compute all of the values $d(0, i, j)$ for every $1 \leq i \leq j \leq n$. This step depends on the model used. Then use the definition (2.5) to compute the values $d(1, 0, j)$ for every possible j, then continue with d(2,0,j) and so on up to the upper bound for the number of breakpoints.

It is trivial to modify the algorithm to find not only the value of d(k,0,j), but also the position of the breakpoints that achieved it: For every computed value of d(k,0,j), we will remember also the last breakpoint b that achieved that maximum. It is the last breakpoint in the desired sequence of breakpoints T for d(k,0,j). the previous members of the sequence are the breakpoints optimizing d(k-1, 0, b) or none, if $k = 1$.

The time complexity of the dynamic optimization algorithm is $\mathcal{O}(kn^2)$, where k is the upper bound for the number of breakpoints. Note that this

time complexity does not incorporate the time needed to compute the values of $d(0, i, j)$. The time complexity depends on the model used.

Once the values of (2.6) are known, it is trivial to optimize the equation (2.3).

The values that perform best on any given task and area can be found experimentally.

In the subsequent sections, we will show three different models for computing the log-probability of any segment.

Chapter 3

Baseline method

In the baseline method, a very simple segment model is used. The model is agnostic to the classes. It can model any land cover type, but it does not model changes of land cover type. The model needs only to provide the probabilities P_S of the data for different segments (i, j), assuming that they contain no change of cover. These values will be then used as an input for the breakpoint detection algorithm proposed in chapter 2.

3.1 Models

Let us have n observations of land cover at times $t_0, t_1, \ldots, t_n$. The times need not to be evenly distributed.

As in [3], we fit a model to each location (pixel) individually. The model fits the values in all bands of the satellite image in the given location in relation to time. Let us call the vector of values measured in time t in some particular location $Y(t)$. The model can be then written as:

$$Y(t) = S(t) + T(t) + e(t). \tag{3.1}$$

The model consists of the following components:

- $T(t)$: trend component representing linear gradual changes over time
- $S(t)$: seasonal component representing periodic annual changes caused by the change of the season
- $e(t)$: the remainder component, assumed to be noise sampled from normal distribution.

We assume that $T(t)$ is a linear function.

The seasonal component S_t represents the periodic changes in the measured values $Y(t)$ caused by the changes of the season. Let us split the year to p equally long intervals. We will call each of these intervals a season, each of the seasons has a number from 0 to $p - 1$. Define function $s(t)$ that assigns the number of the appropriate season to t.

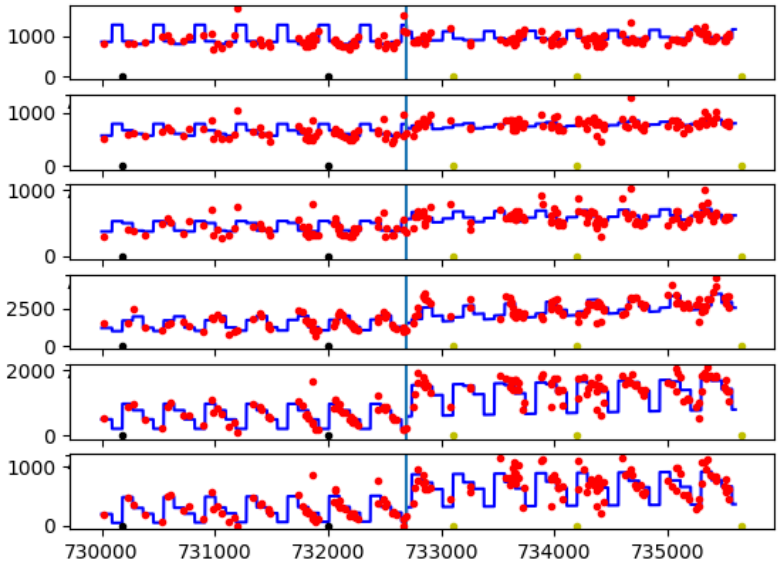


Figure 3.1: sequence of measurements, with a correctly recognized land cover type change detected. In red are the observed values for the six bands in the data, in blue are the models that were fit to the individual segments. Notice the periodic seasonal changes and the linear trends apparent from the fitted models. The black and yellow dots are the reference classifications in the five reference dates, with the colors meaning forest and clear-cut respectively.

Then $S(t)$ is a function of $s(t)$ on any segment.

$$\forall t : S(t) = S(t) = f(s(t)). \tag{3.2}$$

where $f(j)$ is a function that assigns a vector of the seasonal change to every season j.

The remainder component $e(t)$ is assumed to be sampled from normal distribution with zero mean and fixed variance σ^2. We will assume that this variance is common for all modeled classes.

An example of a sequence, where such model was used to detect a break-point, is on image 3.1.

3.2 Fitting the model

3.2.1 Segment models

We will need to efficiently compute estimates of the log-probabilities $P(i, j)$ for every interval in the seqeuence. If the parameters of the trend, seasonal and remainder component are known, then we can define the expected value for any time t as $\hat{y}(t) = T(t) + S(t)$

the probability density to observe a sequence of measurements y is:

$$P_S(y) = \prod_t \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y(t)-\hat{y}(t))^2}{\sigma^2}} \right). \tag{3.3}$$

We are interested in the log-probabilities:

$$\begin{aligned} \log(P_S(y)) &= \sum_{i=1}^{n} \left(-\frac{1}{2}\log(2\pi\sigma^2) - \frac{(y(t)-\hat{y}(t))^2}{\sigma^2} \right) = \\ &= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\sum_t (y(t)-\hat{y}(t))^2. \end{aligned} \tag{3.4}$$

Let us compute the sums of squared residuals:

$$\sum_t (y(t)-\hat{y}(t))^2. \tag{3.5}$$

Let us rewrite $S(t)$ as a linear function of vector $e_{s(t)}$, which is a vector that has all components zero except for the $s(t)$-th component, where $s(t)$ is the number of the season. Then $S(t) = Se_{s(t)}$, where S is a matrix such that j-th column of S is the seasonal change of Y for the season j.

Then the function $T(t) + S(t)$ is a linear function of the vector $X(t)$ and there is a matrix A such that $T(t) + S(t) = A[t, 1, e_{s(t)}]$.

We can construct a matrix $X \in \mathbb{R}^{n,p+2}$ such that i-th row of X is $X(t_i)$.

The matrix has linearly dependent columns, specifically the second column (column of ones) can be expressed as a sum of the columns indicating the season. Thus omitting the column of ones will not decrease the accuracy of the model.

Let us do that and redefine $X(t)$:

$$X(t) = [t, e_{s(t)}]. \tag{3.6}$$

This makes $X \in \mathbb{R}^{n,p+1}$.

Then, we can simply find $A \in \mathbb{R}^{l,p+1}$ minimizing the sum of squared residuals, where l is the length of the vectors $Y(t)$ (the number of bands in the satellite photographs) and p is the number of seasons.

$$SSR(s_i) = \min_{A_i} \sum_{t \in s_i} \|A_i X(t) - Y_i(t)\|^2. \tag{3.7}$$

This is easy to do as $Y_i(t)$ are known and $X(t)$ can be constructed directly from t. The optimal value can be found by solving the equation $X^T XA = X^T Y$, which takes $\mathcal{O}(np(p+l))$ assuming $n \gg p, l$.

Note that this method can be used even if some measurements are missing (for example if some measurements were excluded because of occlusion of the land by clouds).

The naive computation of these values has time complexity $\mathcal{O}(p(p+l)n^3)$. Computing the sums of squared residuals can be done more efficiently than just by computing the values for every interval separately. An efficient algorithm, that computes the sum of squared residuals iteratively, is proposed

in [1]. With the iterative algorithm, computing the values for all intervals has time complexity $\mathcal{O}(n^2p(p+l))$.

With the SSR computed for each interval, let us compute the log-probability densities.

All that remains to compute them is the estimate of σ^2 - the variance of the random noise. If we knew the true number of breakpoints and also the segmentation of the sequence optimal with that number of breakpoints, we could find σ^2 directly as the average of squared residuals. Notice that if we had an optimal segmentation of the sequence with a higher number of breakpoints than the true number, the average of squared residuals would still be affected only by the remainder component, so we could still use it to compute σ^2.

If we fix some number of breakpoints k higher than the maximum expected number of breakpoints, then we can find the segmentation by maximizing:

$$\begin{aligned} &\arg\max_{T:|T|=k} \sum_{s\in S(T)} \log(P_S(x_s)) = \\ &\arg\min_{T:|T|=k} \sum_{s\in S(T)} \sum_{t} (y(t)-\hat{y}(t))^2. \end{aligned} \tag{3.8}$$

This way, we can find the optimal segmentation without the knowledge of σ^2 and compute it as the average of squared residuals in the optimal segmentation.

With the σ estimated and the sums of squared residuals known for any interval, it is straightforward to compute the log-probability density (3.4).

3.3 Segment classification

The linear models described above can be fitted to any any data and used to recognize breakpoints in unsupervised manner. This is an advantage over the other proposed models, that require classified training data. However, the linear model itself does not provide any information about the segment classes, such as forest or clear-cut.

To assign labels to the segments, we use a classifier that classifies models based on their parameters.

3.3.1 Describing the models

To classify the models, we first need to describe each segment model by some vector of its features. The segment model described above is fully described by it's matrix A. Let us use the values in matrix A as the features used for the classification of the models.

3.3.2 Obtain the training labels

To train the classifier, we need some ground truth labels. We will show a method to assign labels to individual segments using labeling with lower

temporal resolution than we use. Figure 3.1 shows an example of such situation. The breakpoint is found and the reference labels are available.

Let us have labelings for each location covering sparsely the whole time period that we are studying.

We will say that a breakpoint is confirmed by the reference, if the last reference label before the breakpoint is of different class than the first reference label after the breakpoint.

We will say that the model is strictly confirmed by the reference, if all of the following conditions hold:

1. every segment in the model contains at least one reference label
2. all reference labels in any segment are of the same class
3. every breakpoint of the model is confirmed by the reference.

Segments of models that are strictly confirmed by the reference can be unambiguously labeled. We can use the features of the models of every segment with their labels to train the classifier. We used a linear KNN with as the classifier.

3.3.3 Down-scaling the labeling from higher resolution

If the labeling is obtained by down-scaling from higher resolution, for every location, there are n labeled high-resolution pixels corresponding to that location. If k of them are labeled as class c, we will interpret the value k/n as the probability that the location is of class c.

There are two possible approaches to validate model with such data. The simplest approach is to choose the most likely label for every location separately, fix them and do the validation with fixed labels as described above.

Another method is to use all possible reference labels (all labeling with non-zero likelihood). We will say that a model is softly confirmed by reference, if there is any labeling with non-zero likelihood that confirms the model.

The second method is less strict - it is more likely to confirm a model. However, it is more robust to sub-pixel errors in alignment of the reference data.

3.4 Results

We tested the method on one particular region in Czech Republic. We divided the studied region in two parts, one used to tune the parameters of the breakpoint estimator and to train the segment classifier, the other to evaluate the accuracy.

We used measurements from the year 2000 to 2015 obtained from Landsat[5] and selected a grid of 100×100 measured locations (pixels of the satellite images). The measurements, where the land was occluded by clouds, were masked by a different algorithm prior to the application of the breakpoint detector. We used all six bands of the satellite measurements.

Five reference images labelled by an expert were used to train the recognizer to recognize three classes: class 0: forest, class 1: growing forest, class 2: forest. The reference labelings were obtained by labeling aerial photographs in high resolution and georeferencing them to the satellite measurements. The reference photographs were taken in the years 2000, 2005, 2008, 2011 and 2015.

3.4.1 Variables

The following parameters of the algorithm needed to be tuned: The number of seasons and penalties for the numbers of breakpoints. Two parts of the recognizer were tested. The first part is the breakpoint estimator, the second part is the segment classifier.

To measure the quality of the breakpoint estimator, two alternative measures can be used. *Consistency* is the ratio of the models that were strictly confirmed by the reference. *Soft consistency* is the ratio of the models that were softly confirmed by the reference.

The classifier was trained from a training dataset and applied to a testing dataset. The classifier labeled every segment in every location in the testing dataset. We will say that the classifier was correct on a location if the labeling that it provided matched all of the reference labelings.

The ratio of the locations where the classifier was correct is the *classification accuracy* of the classifier.

The classifier was trained and applied only on the models consistent with the reference. So a combined measure, the product of classification accuracy and consistency, is the most informative about the recognizer quality. This value is the probability that the recognizer detects breakpoints and segments consistent with the reference. We will call this value the *total accuracy*.

3.4.2 Grid search

We compared the consistency and classifier accuracy of the models with various values of the parameters using the grid-search method. The following parameters were examined.

- number of seasons p: $1, 2, \ldots, 10$.
- breakpoint penalty q: $60, 100, 140, \ldots, 600$.

The penalty for k breakpoint is $q_k = qk$.

Moreover, the parameters were used in two different setups: One with the trend component of the model, $T(t)$, as described in section 3.1, and the other without the trend component, for example with seasonal and remainder component only.

Figure 3.2 shows the performance of the recognizer with different parameters, if fitting both seasonal and trend component. The plot suggests that the total accuracy grows slightly with growing number of seasons, but the effect is almost negligible. The best total accuracy was measured with model

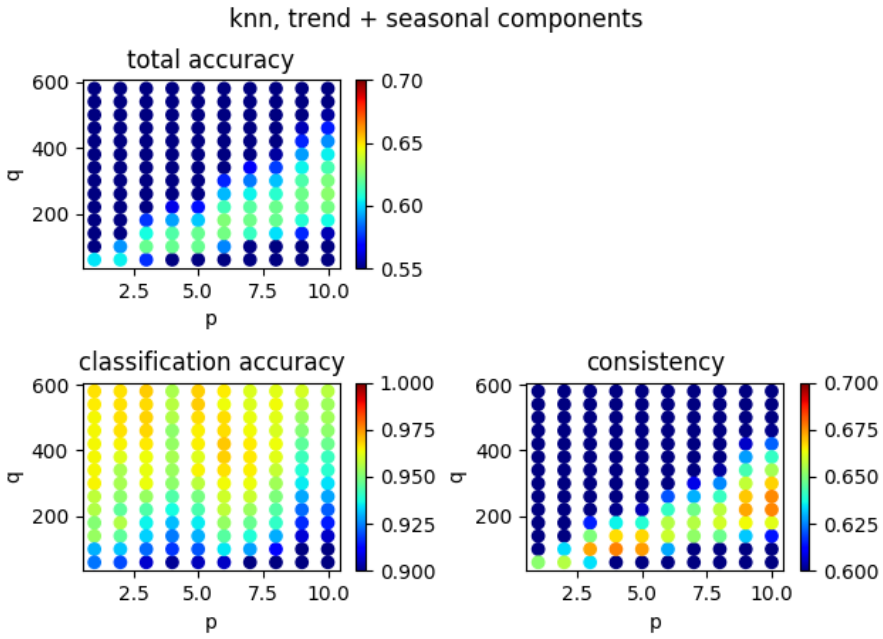


Figure 3.2: recognizer accuracy if using both trend and seasonal components. Models with different number of seasons have similar total accuracy, if the breakpoint penalty is properly adjusted.

using parameters $p = 10$, $q = 260$. The total accuracy was 62.7%. That is a slight improvement over $p = 3$, which had accuracy of 62.1% or $p = 1$ with accuracy 60.2%

Figure 3.3 shows the results of the models that did not use the trend component. It outperformed the more complex model both in breakpoint recognition (consistency) and in segment classification accuracy. The models without the trend component also perform slightly better with higher values of the parameter p, the best result was achieved with 9 seasons ($p = 9$) and the penalty $q = 580$. The accuracy with these parameters was 68%. But even with p=1 (a model that fits a constant function to every segment), the total accuracy with suitable q was 66.6%.

3.4.3 Soft validation results

The consistency score in all of the setups mentioned was below 70%. That was so low mainly because of locations on the boundary of a forest, where the labeling was not clearly determined. To assess the recognizer performance better, the soft confirmation method can be used.

The best performing model (with no trend component, 9 seasons and $q = 580$), achieved soft consistency of 94.5% and classification accuracy (among the segments of the softly confirmed models) of 95.2%, which is together a total accuracy of 89,9%.

The much simpler and faster model, with no trend component, 1 season

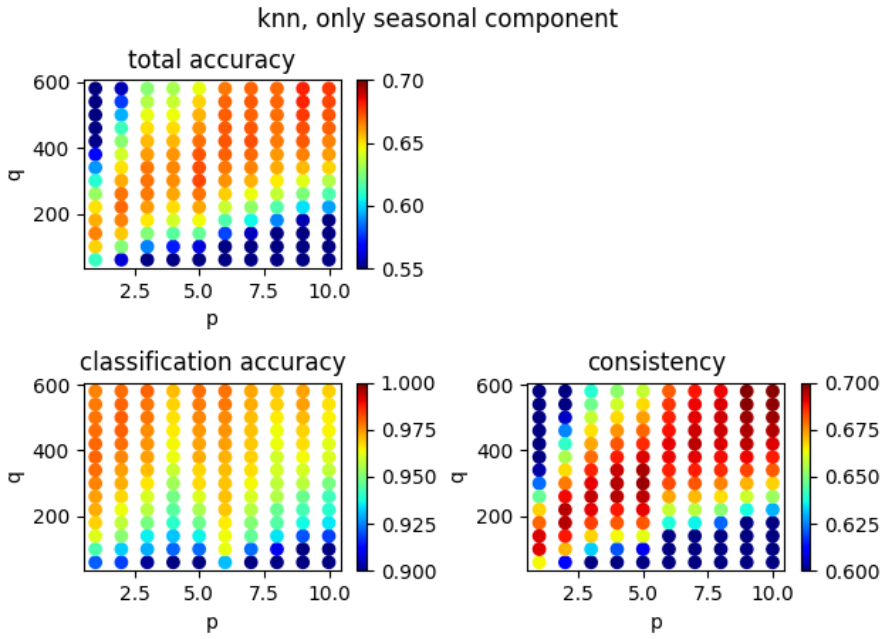


Figure 3.3: models with different number of seasons have similar total accuracy, if the breakpoint penalty is properly adjusted. Not modeling the trend component improved the accuracy of the model.

and $q = 140$ achieved soft consistency 93.4%, classification accuracy 93.5% and total accuracy with soft validation of 87.3%.

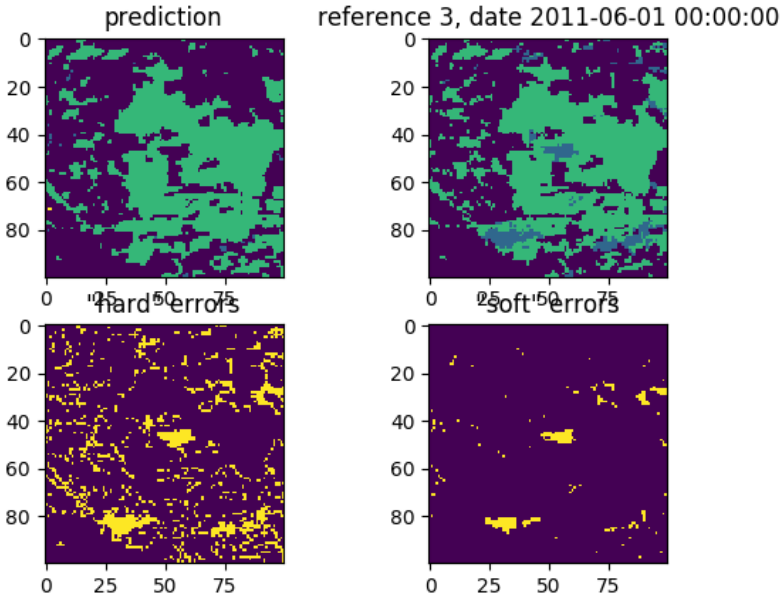


Figure 3.4: Classes detected based on the linear models for the date June 1st 2011 (top-left), compared to the reference classification, obtained from a human classification of an image in higher resolution. Clear-cut in green, growing forest in blue and forest in dark purple. Bottom left shows where the recognized class differs from the most common subpixel class in higher resolution. Bottom right shows where the classification differs from all of the subpixel classes.

Chapter 4

Combination of Hidden Markov Models

The breakpoint detection algorithm described in section 3 consists of two steps. The first step computes the likelihood of any temporal interval in a model that does not allow breakpoints. The second step finds the most likely segmentation.

We propose an improvement of this algorithm that can be used if the classes are known in advance and if we have training data for them. The breakpoint detection works the same way as described above, only a different approach to get the likelihood of an interval is used.

A model is trained and fitted for every class separately. The likelihood of an interval is the likelihood of the most likely model (the model of the most likely class on that interval). Note that this approach provides not only likelihood for any segment, but also a classification. The likelihoods obtained this way are then used to find the most likely segmentation.

4.1 Models

For the linear model, we used a sequence of vectors x_i describing the observed measurements (pixel values in six bands) for a given location and the vector y_i describing the season or time. For HMM, we need only a single sequence. A concatenation of the vector describing the measurements and the vector describing the season is used. From now on we will call such vector x_i. It will be used for both training and fitting the models.

Some measurements in the sequence are missing because of the occlusion by clouds. Such measurements can simply be omitted before training or fitting the model, so that the model deals only with relevant data.

4.1.1 Training

To train the Hidden Markov Models for some class (for example forest), sequences x that are known to belong to this class are needed. We used the baseline method to detect breakpoints and classify segments. Only the locations that were classified consistently with the reference were used. Strict validation was used for the consistency check, as described in chapter 3. From

each validated segmentation, the segments were used as training data for the appropriate HMM.

Training data can be obtained even from the locations where breakpoints were not detected consistently with the reference. We can assume that if two reference classifications are the same class for some location, then there is no breakpoint between them and the inteval between them can be used as a training segment for the reference class. The interval between reference classifications with different classes can have breakpoint anywhere in it and we cannot assume anything about that interval.

4.1.2 Model fitting

Once the separate models for the classes are trained, the models are fitted on every interval. We need to find the likelihood of every model on every possible interval in the sequence. We show an efficient method to do that.

We need to compute the probability of the sequence $x_{i,\dots,j} = x_i, x_{i+1}, \dots, x_j$ for every interval (i, j) for every model (each class has one model).

With some of the models fixed, for every starting time i, we can iteratively compute the likelihoods of all the intervals starting at i. We will do a forward pass with the model on the sequence $x_{i,\dots,j}$. Then the forward message in the forward pass $\varphi(k, s)$ represents the probability of reaching the state s on x_{i+k}. If we sum over all states, we get the probability of the subsequence $x_{i,\dots,j}$:

$$P((i, j)) = \sum_s \varphi(k, s). \tag{4.1}$$

This implies that computing the probabilities of every subsequence of x starting in i, is asymptotically not harder than just computing the probability of the sequence $x_i, \dots, x_n$, which requires the same forward pass.

The time complexity to compute the probability of all intervals (i, j) for some model is $\mathcal{O}(n^2m^2)$, where m is the number of states in the model. Obviously this needs to be done for every model.

Once the likelihoods for every interval for every class model are known, we apply the dynamic algorithm for breakpoint detection described in chapter 2. The likelihood of an interval will be the likelihood of the most likely class model. The classification of the resulting segmentation will be the classes of the most likely models for the respective segments.

4.2 Results

The algorithm was applied on the same dataset as the baseline method. We used hmm with 12 hidden states for every of the three recognized classes. The input features x contained the measured data and encoded season.

The measured data were for any time and location six vaues from the six bands of the photographs. The inputs where the measurements were not informative because of cloud occlusion,were simply removed from the sequence.

The encoded season were four binary values indicating which quarter of the year was when that measurement was taken.

The total accuracy improved to 96% with soft validation and to 75.4% with strict validation. The errors can be attributed mostly to locations containing the class "growing forest," which was never detected by this algorithm and to locations on the boundaries of forests, see illustration 4.1.

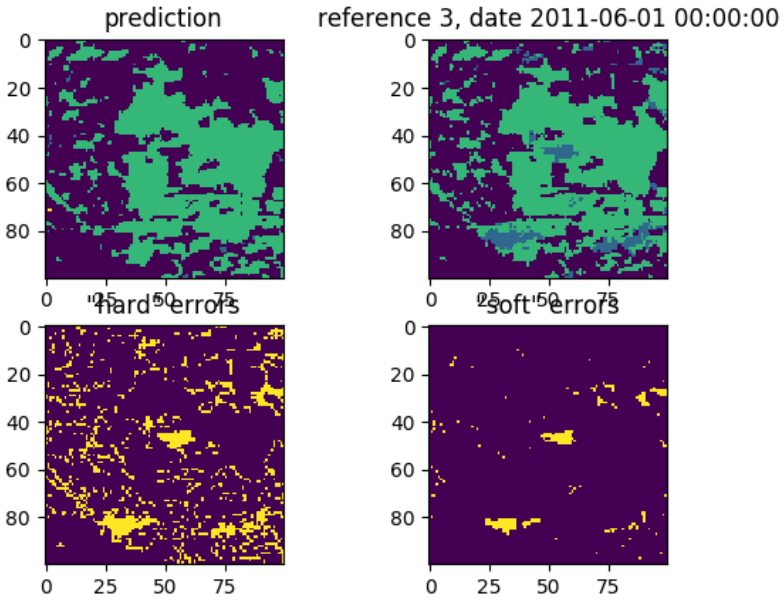


Figure 4.1: Detected classes for the date June 1st 2011 (top-left), compared to the reference classification, obtained from a human classification of an image in higher resolution. Clear-cut in green, growing forest in blue and forest in dark purple. Bottom left shows where the recognized class differs from the most common subpixel class in higher resolution. Bottom right shows where the classification differs from all of the subpixel classes.

Chapter 5

Recurrent Neural Network

In this section we propose an alternative method. The method is based on a recurrent neural network. RNNs have been used successfully for satellite data in [6], [7], [8]. We propose a single recurrent neural network that takes a sequence of x_i, describing the measured values for a location together with the encoded season. The network is trained to output classification for each of the vectors.

The missing data were not omitted when training and fitting the neural network. They were replaced with zeros. To disambiguate between missing values and pixels where the zeros were truly measured, another feature can be added to the input feature vector indicating that the measurement is removed for x_i.

5.1 Architecture of the network

From the total of 10000 locations (pixels), We used 100 randomly selected locations for validation, the other were used for training.

The network consists of an input filter, rectified linear unit, gated recurrent unit for the forward pass, gated recurrent unit for the backward pass and an output filter.

Every vector in the input sequence is processed with input filter, and continues to both forward GRU and backward GRU. The outputs of both forward GRU and backward GRU are concatenated and passed to the output filter.

The input filter consists of a fully connected linear layer and a ReLU. The output filter consists of a fully connected linear layer and softmax.

5.1.1 Training

The classification used for training were the fully and partially classified sequences obtained from two sources: first, the segmentations detected by the breakpoint detector with the linear model, if they were found to be strictly consistent with the reference. Second, on locations, where such segmentation is not strictly consistent, a partial classification obtained from the aerial

photographs is used. This is the same approach that was used to get the training data for HMMs. It described in more detail in section 4.1.1.

The difference from the HMM training is that while HMMs were trained as separate models for the individual classes, here we train a single network modelling the whole sequence including the breakpoints. Note that in some sequences in the training dataset, some of the classifications are missing. We propose a loss function that handles such inputs.

5.1.2 Architecture of the network

Recurrent neural networks are a type of neural network that can process sequences of inputs and update some inner state vector. The state vector can be then sent to a classifier that outputs some class prediction for every inner state, which is a classification of the corresponding network input. This way, the classification is based not only on the given input, but also on the preceeding inputs in the sequence. Information from them is passed via the inner state.

A bidirectional RNN does this pass twice: once forward and once backward. Each of the two passes maintains a separate hidden state. To classify an input, one can concatenate the corresponding inner states from the backward and forward pass.

Gated recurrent unit (GRU) was used as the recurrent cell for both the forward and backward pass. The size of the hidden state for both forward and backward pass was 64.

To improve the expressiveness of the network, an additional filter consisting of a fully connected linear layer and ReLU is applied to the input prior to passing it to GRU. The filter outputs a vector of size 12. The classifier consists of a linear layer and softmax.

The architecture of the network is depicted on the figure 5.1

5.1.3 Loss function

A special loss function is used to enforce correct classification only in positions where the reference classification is known and to enforce low number of breakpoints at the same time.

The loss function can be written as

$$l(y', y) = \sum_{i=1}^{n} \mathbb{1}(y_i \neq unknown) H(y'_i, y_i) + q \cdot \sum_{i=1}^{n-1} H(y'_i, y'_{i+1}). \tag{5.1}$$

where y' is the sequence of network outputs, y are the correct classifications, q is a metaparameter weighing the cost of a breakpoint and $H(x, y)$ is the cross entropy.

The first sum in the loss function enforces that the network predicts consistently with the training data. The second sum enforces that the adjacent predictions are similar. This reduces the number of breakpoints recognized in the sequence.

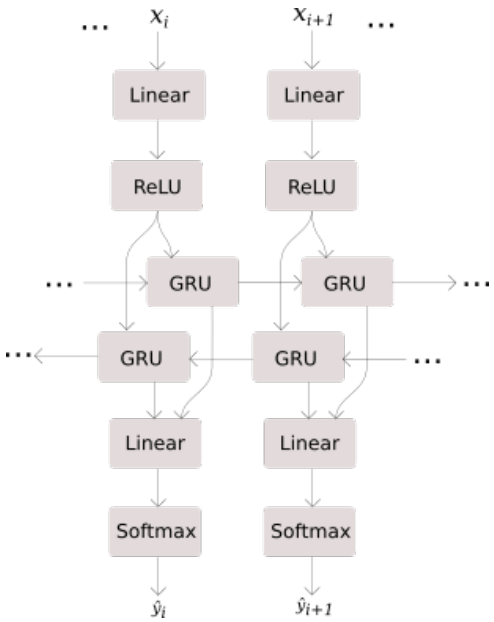


Figure 5.1: The architecture of the RNN

5.1.4 Predicting

The network, given an input sequence, outputs a sequence of values that can be interpretted as likelihoods for the respective classes. Although the training loss function optimizes that the predictions are consistent, in some cases it still predicted breakpoints that were not a true change of land cover, but only noise on the forest boundaries caused by the improper alignment of the data.

To obtain a temporal segmentation of the sequence, that enforces low number of breakpoints better, we will use the same breakpoint detection algorithm as in the previously described methods. The likelihood of an interval for a class will be the product of the likelihoods of that class in the interval.

5.1.5 Results

The evaluation process was the same as was used for the other approaches. The accuracy with soft validation was 97.6%. Accuracy with the scrict validation was 78.9%.

Figure 5.2 shows a visualization of the method. Notice that there are obviously much less errors compared to the approach with multiple HMMs. Also note that this algorithm was able to recognize the class "growing forest", which was not detected by the other approaches.

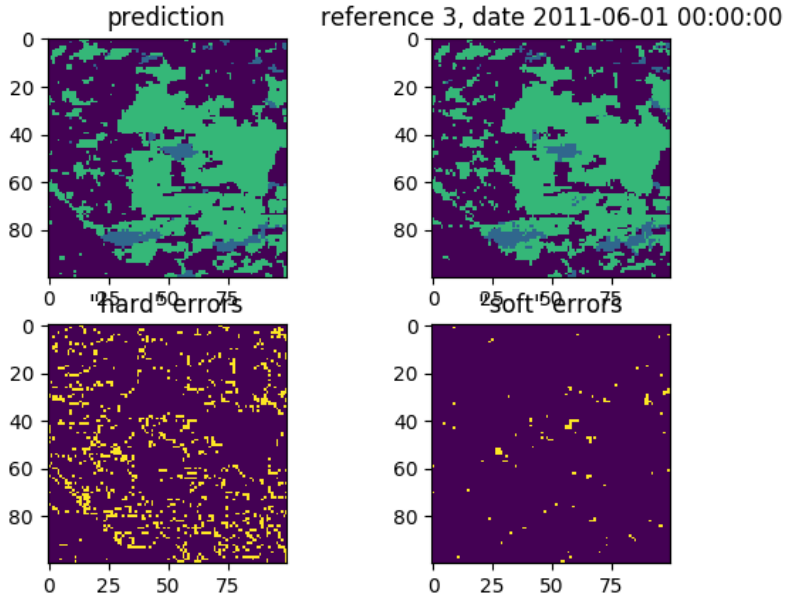


Figure 5.2: compilation of classes detected by rnn for the date June 1st 2011 (top-left), compared to the reference classification, obtained from a human classification of an image in higher resolution. Clear-cut in green, growing forest in blue and forest in dark purple. Bottom left shows where the recognized class differs from the most common subpixel class in higher resolution. Bottom right shows where the classification differs from all of the subpixel classes.

Chapter 6

Conclusion

6.1 Comparison of the models

On the testing data that we used, the recurrent neural network performed with the highest accuracy from the three examined approaches, with comparable training and execution times.

6.2 Unsolved problems

For the algorithms to work sufficiently, some challenges still need to be overcome. One of the problems to solve is the spatial alignment of the measurements. Although the measurements from Landsat were georeferenced, the precission is not good enough for the proposed algorithms to work reliably on the boundaries of different land covers. Some further preprocessing of the sequence, that would ensure that all of the images are consistently aligned, would be necessary.

Second, the individual images in the six bands of the measurements had inconsistent colors, each of them was tinted in different shade. This effect could be either removed by normalizing the data or by modelling the tint.

However none of those problems were in the scope of this work.

6.3 Attributions

The following third-party software was used to implement the mentioned approaches:

- hmmlearn - a python open-source library for HMM
- tensorflow - a python open-source library for neaural networks
- implementation of an example RNN was kindly provided by Jindrich Libovicky.

Bibliography

[1] Brown et al. *Techniques for Testing the Constancy of Regression Relationships Over Time* Journal of the Royal Statistical Society: Series B (Methodological), 1975

[2] Jushan Bai and Pierre Perron *Computation and analysis of multiple structural change models* Journal of applied econometrics, 2003

[3] Verbesselt, Jan, et al. *Detecting trend and seasonal changes in satellite image time series.* Remote sensing of environment, 2010

[4] Drusch, M., et al., *Sentinel-2: ESA's optical high-resolution mission for GMES operational services.* Remote Sensing of Environment, 2012

[5] ,

[6] P. Benedetti et al., *M3Fusion: A Deep Learning Architecture for Multi-Scale/Modal/Temporal satellite data fusion*, arXiv:1803.01945, 2018

[7] D. H. T. Minh, D. Ienco, R. Gaetano, N. Lalande, E. Ndikumana, F. Osman, and P. Maurel, *Deep recurrent neural networks for winter vegetation quality mapping via multitemporal sar sentinel-1,* , IEEE GRSL, vol. Preprint, no. -, pp. –, 2018.

[8] L. Mou, P. Ghamisi, and X. X. Zhu, *Deep recurrent neural networks for hyperspectral image classification*, IEEE TGRS, vol. 55, no. 7, pp. 3639–3655, 2017.