PR: Optimizing the FUSINTER Algorithm

Florian Jaksik, WS 2023/24

Fusinter Discretization

What is Discretization?

- Discretization is the process of transforming continuous into discrete data by constructing intervals.

  E.g. temperature given in Degree Celsius into cold (-inf,10], mild (10,25], hot (25, inf).

- This can be achieved in **supervised** and **unsupervised** ways.

  Examples for Unsupervised:

  > **Domain-Dependent**: E.g. Phases of growing up; Baby (0-3], Child (3,6], School Child (6-10]
  >
  > **Equal-Width**: Divide the range between the minimum and maximum data points into equal intervals; (0,3] , (3,6], (6,9]...
  >
  > **Equal-Frequency**: Divide the range into intervals that have the same amount of data points.

- For unsupervised methods no knowledge of labels is needed.

What is Discretization?

- Supervised methods use knowledge about the labels.
- If the method starts with a single interval over all data points and iteratively splits it, it is called **top-down discretization**.
- If it starts with an interval for each data point and iteratively merges them, it is called **bottom-up discretization**.

  Examples:

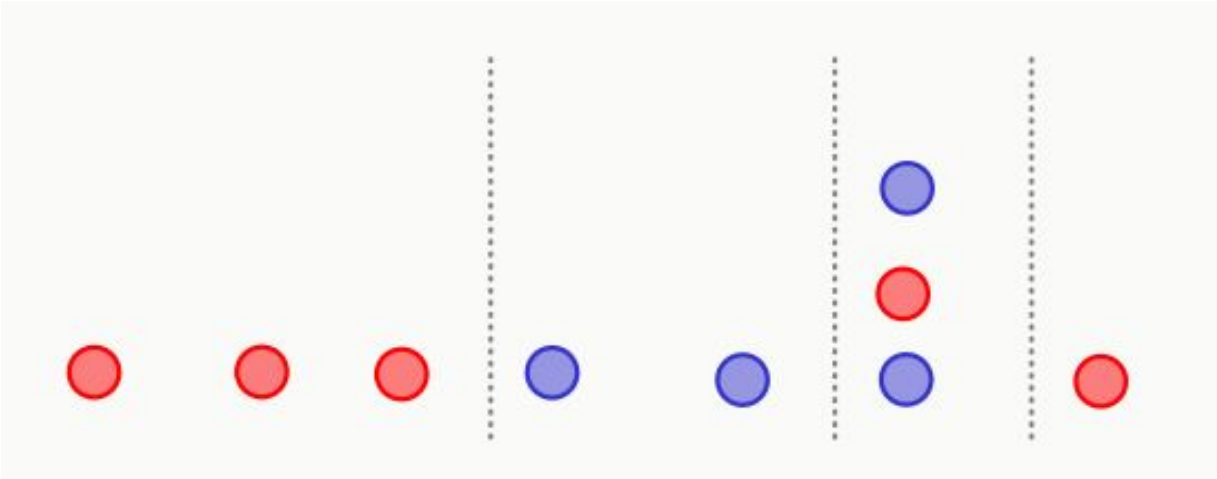
  top-down, supervised: **Entropy Split**

  bottom-up, supervised: **Chi-Merge**

- FUSINTER uses information of the labels and iteratively merges intervals, so it has to be classified as bottom-up, supervised.

FUSINTER: How does it work?

1. The sorted data is sweeped from lowest to highest value. If a run of equal labels ends an interval split point is inserted.
2. If multiple data points of different labels have the same value we close the current interval and open another one until the next data point with a different value is found.



FUSINTER: How does it work?

3. We build a contingency table based on the split points and the labels occurring in each interval.

3	0	1	1
0	2	2	0

4. For all consecutive columns of the table we compute tables where these columns are merged.

3	1	1
2	2	0

3	1	1
0	4	0

3	0	2
0	2	2

FUSINTER: How does it work?

5. With the help of a positive, real valued function $\phi(T)$ we find some split index i such that for the original table T and all merged tables T_k we get

$$i = \arg\max_k \phi(T) - \phi(T_k)$$

6. If the difference for the index i is greater than 0, we remove the i-th interval split and rename the i-th merged table to T and go back to step 3.
7. Else we return the current splits and end the algorithm.

FUSINTER: How are split values calculated?

- In the paper two functions for ϕ are proposed:

$$\text{Shannon's Entropy } = \phi_1(T) := \sum_{j=1}^{k} \Big\{ \alpha \frac{n_j}{n} \Big(-\sum_{i=1}^{m} \frac{n_{ij}+\lambda}{n_j + m\lambda} \log_2 \frac{n_{ij}+\lambda}{n_j+m\lambda} \Big) + (1-\alpha)\frac{m\lambda}{n_j} \Big\}$$

$$\text{Quadratic Entropy } = \phi_2(T) := \sum_{j=1}^{k} \Big\{ \alpha \frac{n_j}{n} \Big(\sum_{i=1}^{m} \Big(\frac{n_{ij}+\lambda}{n_j+m\lambda} \Big) \Big(1 - \frac{n_{ij}+\lambda}{n_j+m\lambda} \Big) \Big) + (1-\alpha)\frac{m\lambda}{n_j} \Big\}$$

- Where the variables are defined as:

k : number of splits in table

n : number of examples in table

m : number of unique labels

n_j : number of examples in j-th column

n_{ij} : value of cell in row i and column j

λ **and** α : tuning parameters

FUSINTER: How do the parameters affect the splits?

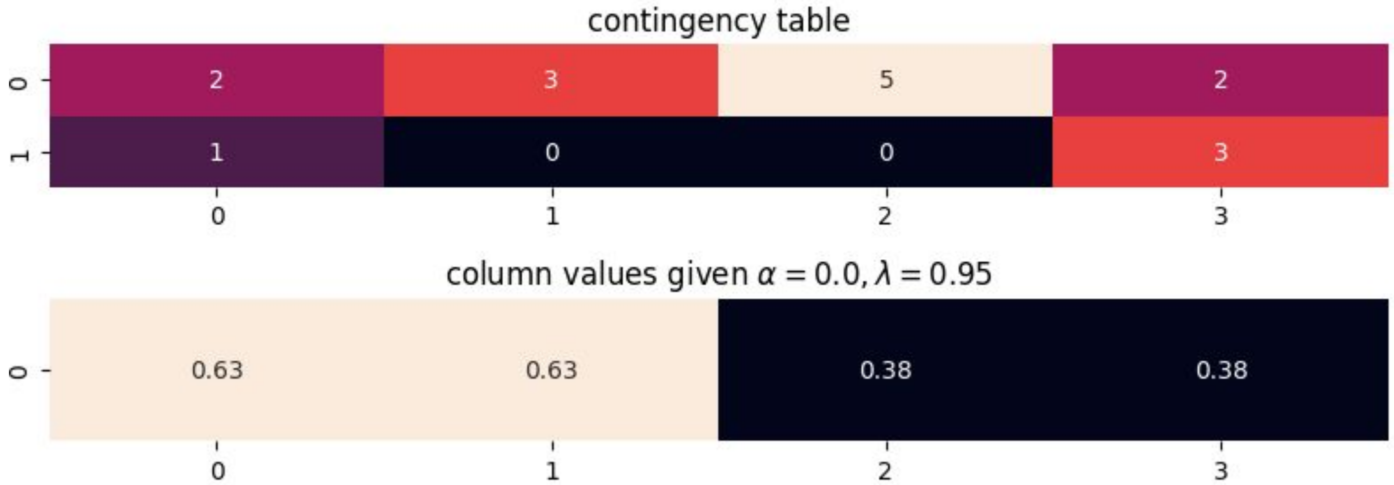
- If we look closer at the Shannon Entropy defined for the algorithm, we find that alpha affects the weighting between the inner summation and some constant value for each column.

$$\text{Shannon's Entropy } = \phi_1(T) := \sum_{j=1}^{k}\left\{\underline{\alpha\frac{n_j}{n}\left(-\sum_{i=1}^{m}\frac{n_{ij}+\lambda}{n_j+m\lambda}\log_2\frac{n_{ij}+\lambda}{n_j+m\lambda}\right)}+\underline{(1-\alpha)\frac{m\lambda}{n_j}}\right\}$$

- So If alpha = 0 the column values should only depend on the number of examples in each column and not on the distribution.
- For alpha in (0,1] the column values are the result of weighting the two terms by alpha and its complement.

FUSINTER: How does alpha affect the column values?

- Here we have 4 columns with the first two having 3 examples and the last two having 5 examples each. For alpha = 0 each of those pairs should have the same value.



contingency table

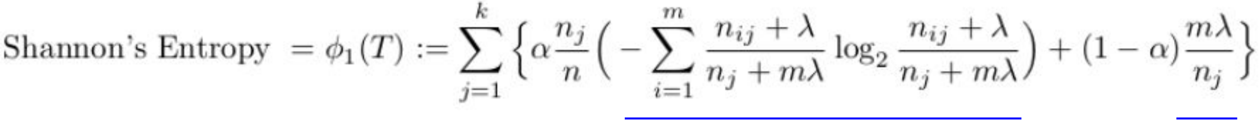
	0	1	2	3
0	2	3	5	2
1	1	0	0	3

column values given $\alpha = 0.0, \lambda = 0.95$

	0	1	2	3
0	0.63	0.63	0.38	0.38

FUSINTER: How does lambda affect the column values?

- The lambda values is used in both terms of the inner sum. Once for Laplace Smoothing and once in the second constant term.

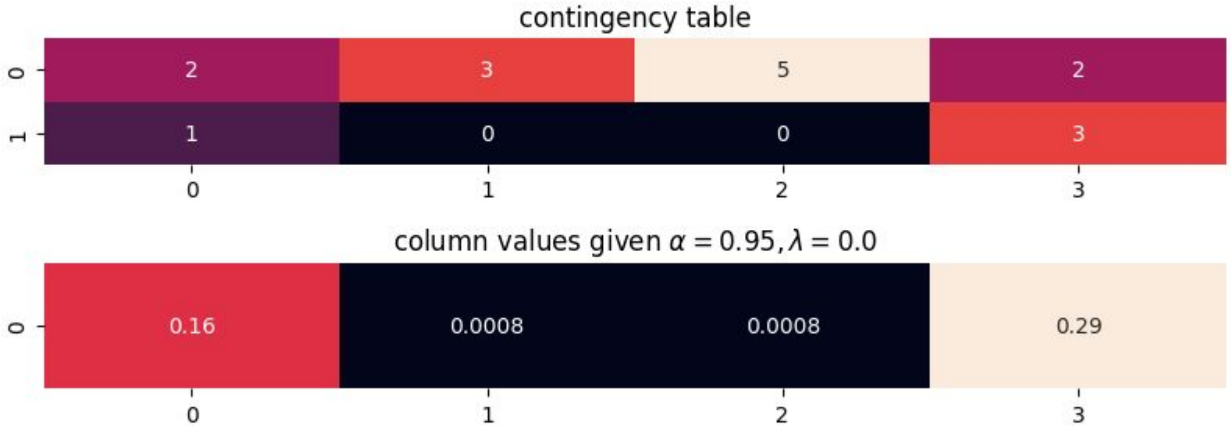


$$\text{Shannon's Entropy } = \phi_1(T) := \sum_{j=1}^{k}\left\{\alpha\frac{n_j}{n}\underline{\left(-\sum_{i=1}^{m}\frac{n_{ij}+\lambda}{n_j+m\lambda}\log_2\frac{n_{ij}+\lambda}{n_j+m\lambda}\right)}+(1-\alpha)\underline{\frac{m\lambda}{n_j}}\right\}$$

- If lambda = 0, the column values are following the classical definition of the Shannon Entropy.
- If lambda goes to infinity the column value will not depend on distribution anymore.

FUSINTER: How does lambda affect the column values?

- With lambda = 0 the more concentrated columns have smaller values and the more spread columns have higher values.



contingency table

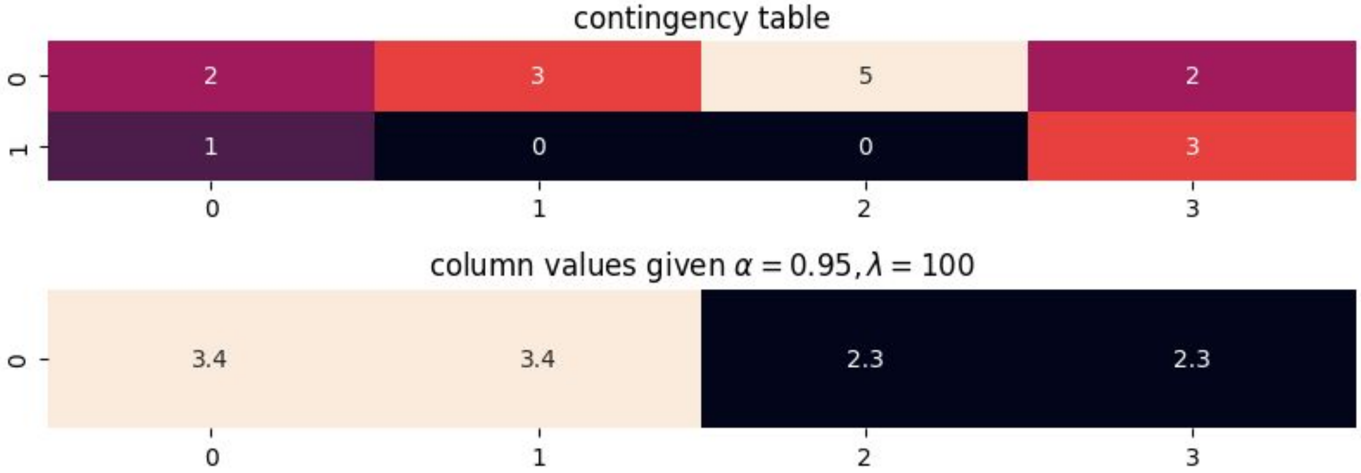
	0	1	2	3
0	2	3	5	2
1	1	0	0	3

column values given $\alpha = 0.95, \lambda = 0.0$

	0	1	2	3
0	0.16	0.0008	0.0008	0.29

FUSINTER: How does lambda affect the column values?

- For lambda = 100 the influence of the distribution of labels in the columns vanishes.



Examples of different alpha and lambda values

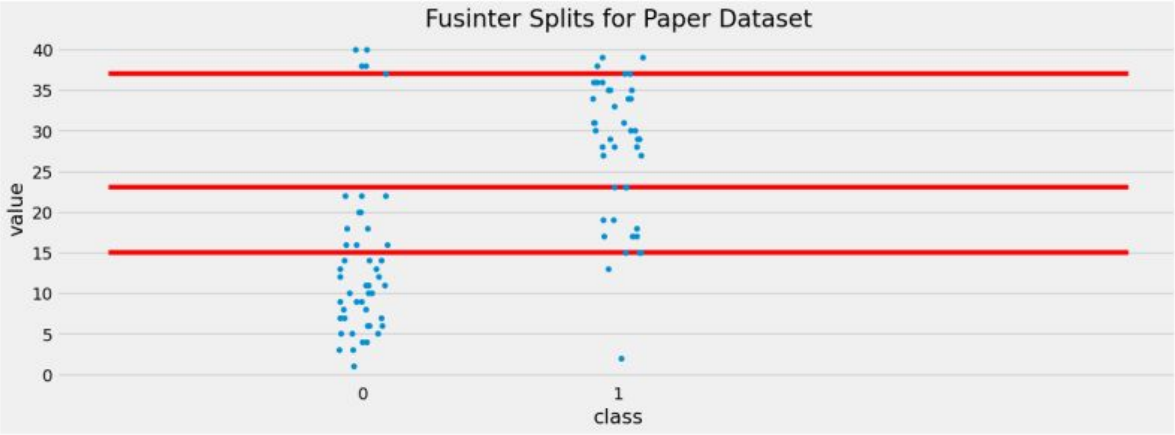


Figure 3: Splits with $lambda = 1$ and $\alpha = 0.95$

Examples of different alpha and lambda values

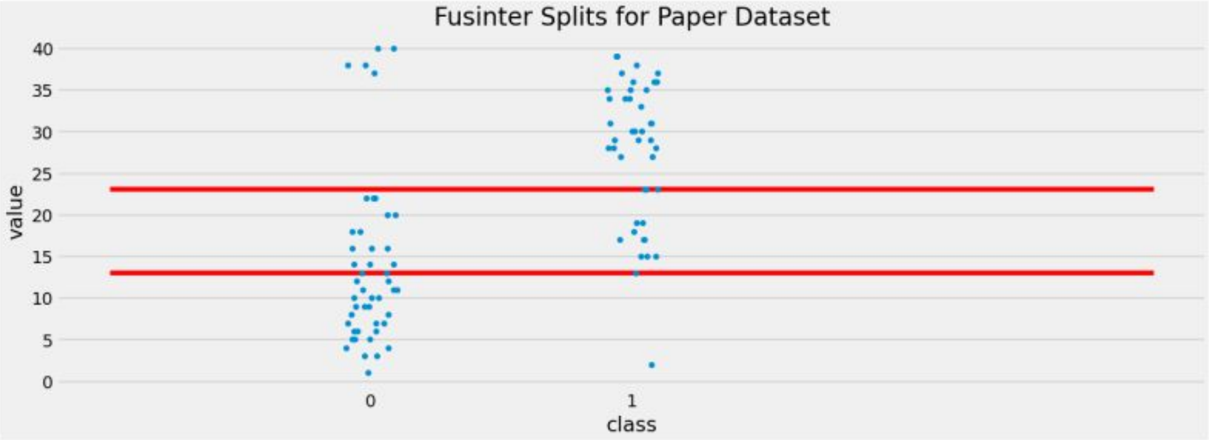


Figure 4: Splits with $lambda = 1$ and $\alpha = 0.85$

Examples of different alpha and lambda values

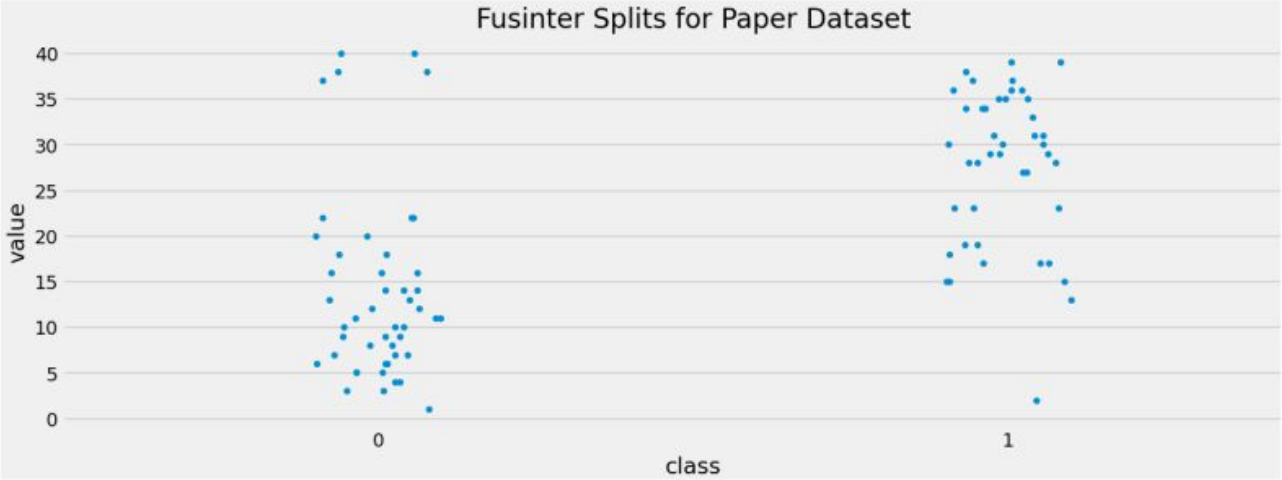


Figure 6: Splits with $lambda = 1$ and $\alpha = 0.25$

Examples of different alpha and lambda values

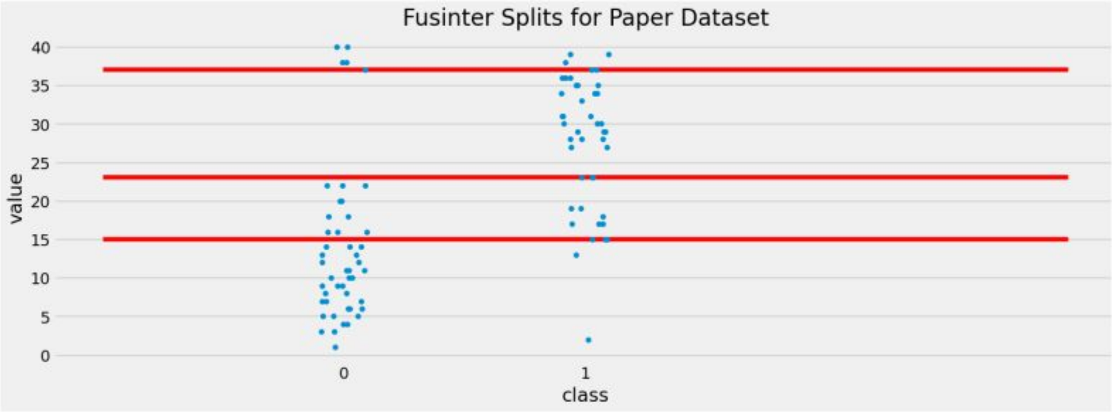


Figure 7: Splits with $lambda = 1$ and $\alpha = 0.95$

Examples of different alpha and lambda values

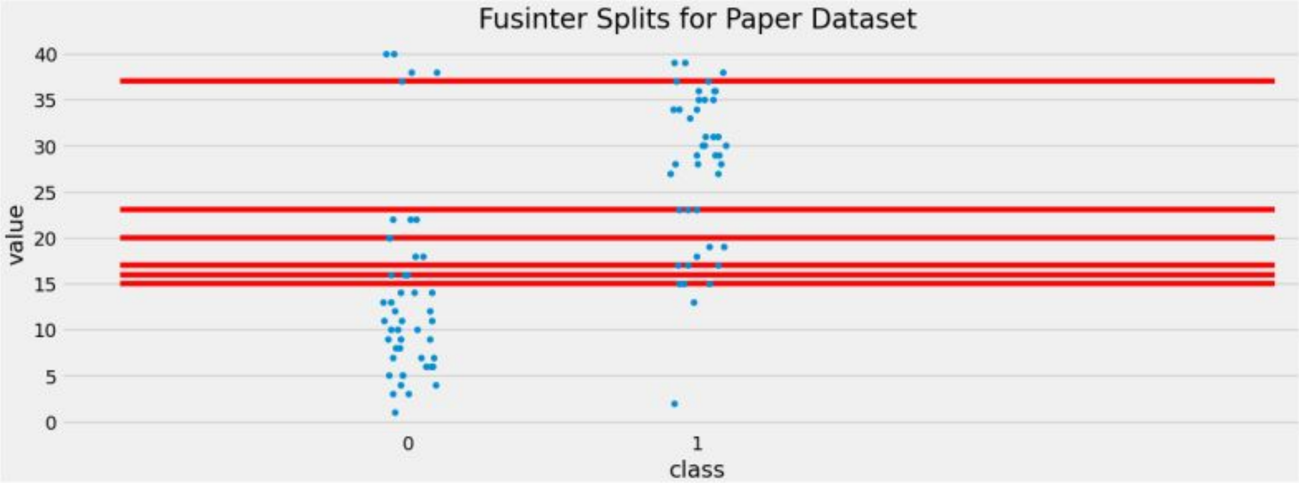


Figure 9: Splits with $lambda = 0.35$ and $\alpha = 0.95$

Examples of different alpha and lambda values

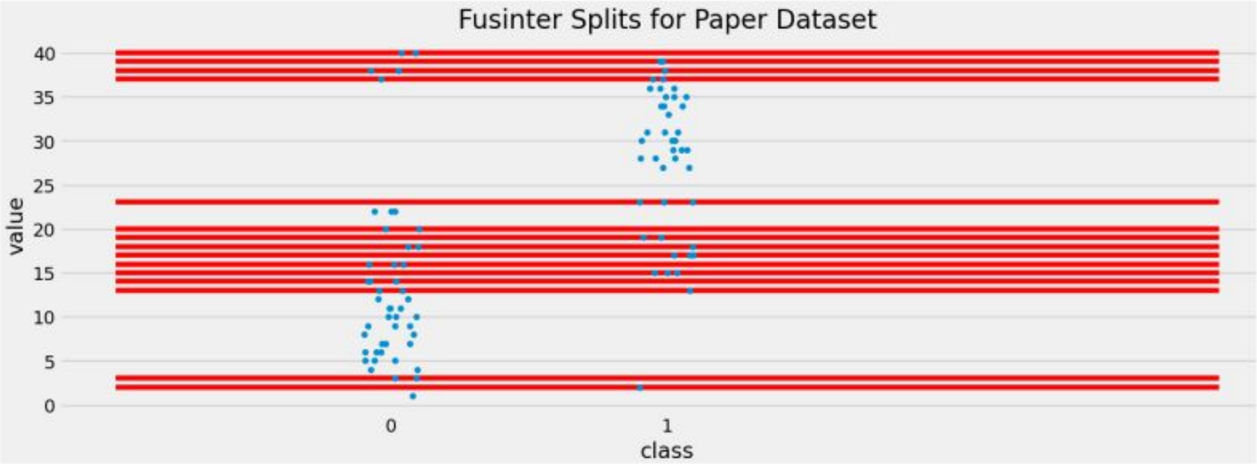


Figure 11: Splits with $lambda = 0.05$ and $\alpha = 0.95$

Interface:

- `FUSINTERDiscretizer(alpha, lambda)` : Initializes the discretizer
- `discretizer.fit(x, y)` : Fits the discretizer to the values x with labels y
- `discretizer.transform(x)` : Discretizes the values x

Implemented Optimizations

Implemented Optimization: V1

- The naive implementation computes

$$i = \arg\max_k \phi(T) - \phi(T_k)$$

  over all possible values of k.

- But a look at the entropy functions reveals that we can store most of outer sum and only compute a new value for the column resulting from the merge.

$$\phi_1(T) := \sum_{j=1}^{k} \left\{ \alpha \frac{n_j}{n} \left(- \sum_{i=1}^{m} \frac{n_{ij}+\lambda}{n_j+m\lambda} \log_2 \frac{n_{ij}+\lambda}{n_j+m\lambda} \right) + (1-\alpha) \frac{m\lambda}{n_j} \right\}$$

Implemented Optimization: V1

Example:

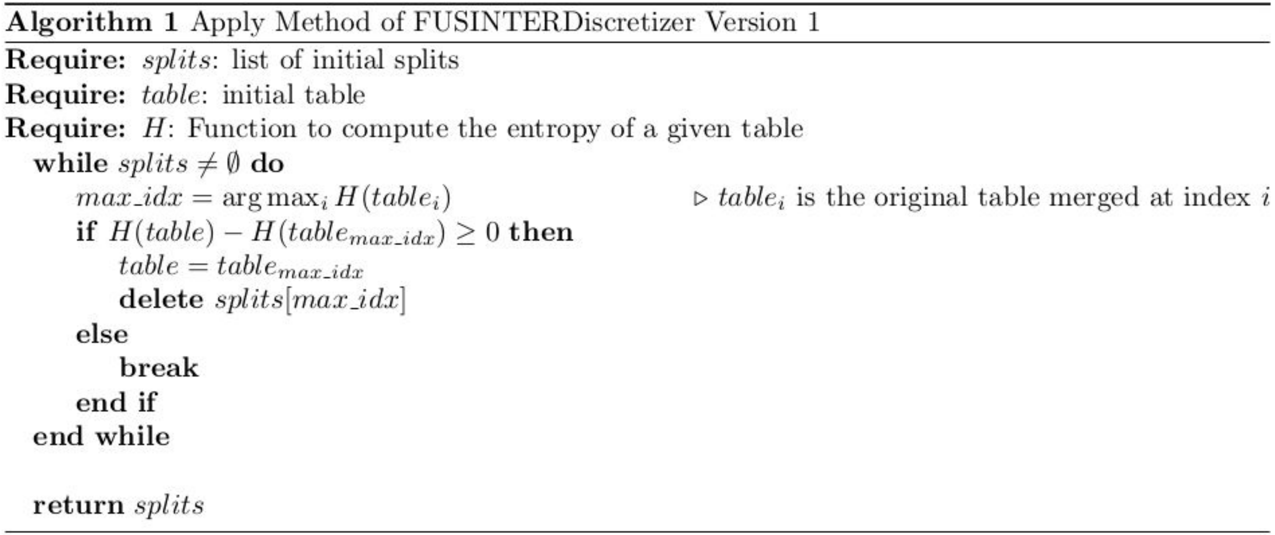
- Let the summands of the outer sum over a contingency table T be

a	b	c	d

- Assume we want to compute the value for a merge of the second and third column: $\phi(T) - \phi(T_2) = a + b + c + d - (a + g(2) + d) = b + c - g(2)$
- Here g(2) is the value of the column created by the merge of the second and third columns of T.
- So it is possible store the column values of T and do this small calculation once for each possible split.

Implemented Optimization: V1

- The naive implementation can be described in the following way:



Algorithm 1 Apply Method of FUSINTERDiscretizer Version 1

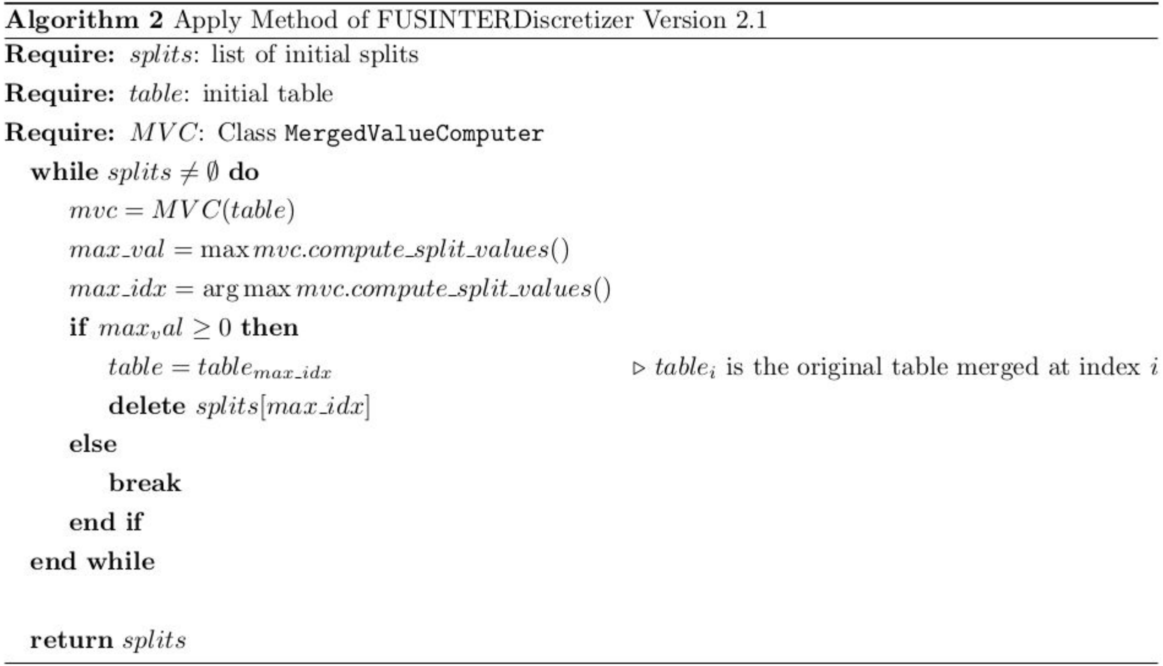
```
Require: $splits$: list of initial splits
Require: $table$: initial table
Require: $H$: Function to compute the entropy of a given table
  while $splits \neq \emptyset$ do
    $max\_idx = \arg\max_i H(table_i)$        ▷ $table_i$ is the original table merged at index $i$
    if $H(table) - H(table_{max\_idx}) \geq 0$ then
      $table = table_{max\_idx}$
      delete $splits[max\_idx]$
    else
      break
    end if
  end while

  return $splits$
```

- Here we compute H once for each split and then remove one split.

Implemented Optimization: V1

- The optimized version used a class named MergeValueComputer to store already computed column values and compute the value of a given merge index.



Algorithm 2 Apply Method of FUSINTERDiscretizer Version 2.1

```
Require: $splits$: list of initial splits
Require: $table$: initial table
Require: $MVC$: Class MergedValueComputer
  while $splits \neq \emptyset$ do
    $mvc = MVC(table)$
    $max\_val = \max mvc.compute\_split\_values()$
    $max\_idx = \arg\max mvc.compute\_split\_values()$
    if $max_val \geq 0$ then
      $table = table_{max\_idx}$                ▷ $table_i$ is the original table merged at index $i$
      delete $splits[max\_idx]$
    else
      break
    end if
  end while

  return $splits$
```

Implemented Optimization: V1

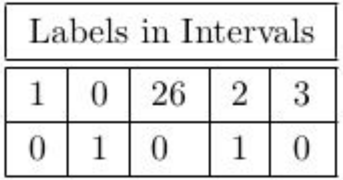
- The main difference between the naive implementation and V1 is that we do not have to create new tables for each split, but we can use the original table T and calculate the values of the merges from T.

Implemented Optimization: V2

- In V1 we computed the split values by iterating over the whole contingency table in each step.
- But most of the values $\phi(T) - \phi(T_i)$ stay the same over each iteration.
- If we look at the values of T and the merged version T_i, it occurs that only the values for i - 1 and i + 1 change values. This can be used to save computation time.
- We introduce the value delta to indicate the difference between the entropy values of the original table and the merged table.

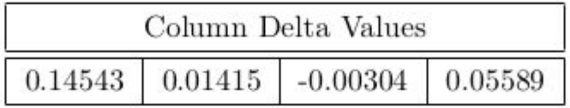
Implemented Optimization: V2

E.g.: Assume the following contingency table



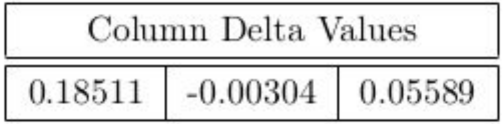
Labels in Intervals				
1	0	26	2	3
0	1	0	1	0

The delta values computed from this table are



Column Delta Values			
0.14543	0.01415	-0.00304	0.05589

In the next iteration we have to merge the first two columns, so the next delta vector is



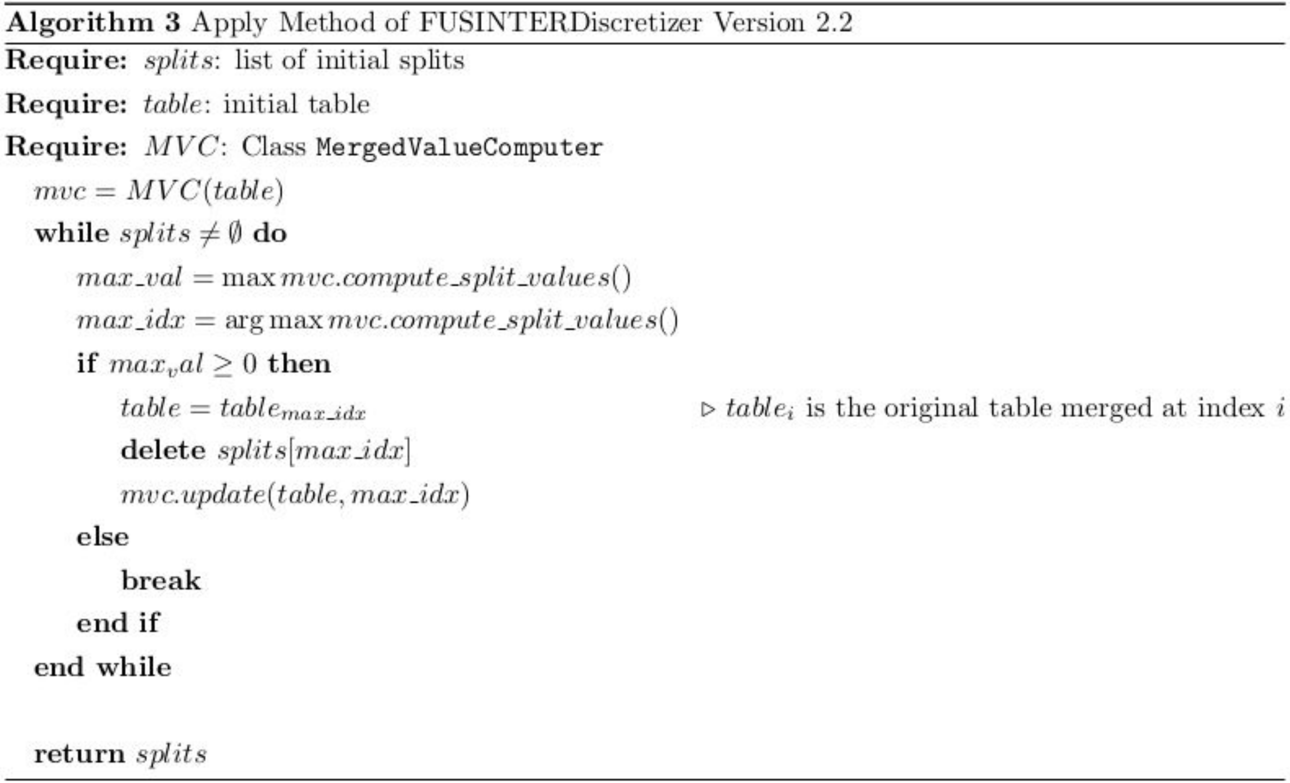
Column Delta Values		
0.18511	-0.00304	0.05589

Only the successor of the merged delta changes.

Implemented Optimization: V2

- I used MergeValueComputer to compute the deltas once in the beginning like for V1 and then use these stored values to compute the updates.
- This only necessitates at most two updates per iteration instead of one update for each column of the table per iteration.
- For this to work the MergeValueComputer has to store the deltas and update them according to the values of the merged consistency table.

Implemented Optimization: V2



Algorithm 3 Apply Method of FUSINTERDiscretizer Version 2.2

```
Require: $splits$: list of initial splits
Require: $table$: initial table
Require: $MVC$: Class MergedValueComputer
  $mvc = MVC(table)$
  while $splits \neq \emptyset$ do
    $max\_val = \max mvc.compute\_split\_values()$
    $max\_idx = \arg\max mvc.compute\_split\_values()$
    if $max_val \geq 0$ then
      $table = table_{max\_idx}$                ▷ $table_i$ is the original table merged at index $i$
      delete $splits[max\_idx]$
      $mvc.update(table, max\_idx)$
    else
      break
    end if
  end while

  return $splits$
```

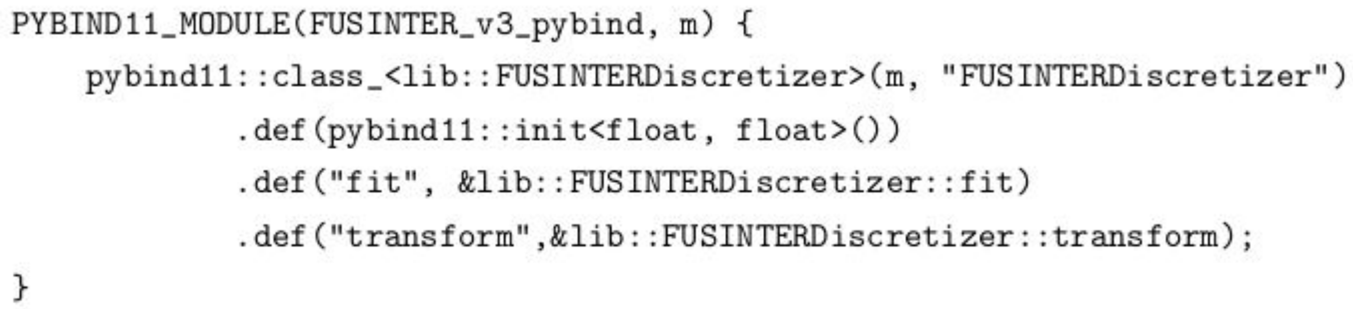
C++ Implementation

C++ Implementation

- Except for C++ idioms the implementation follows V2 of the Python implementation.
- The Python binding is achieved with the help of the Pybind11 library.
- The final output is a `.so` file that can be directly imported and used as a module in Python.

How to make the C++ code callable

- The algorithm itself is implemented in C++
- To create the `.so` file pybind has to know what needs to be compiled and how it should be named. This is done in the pybind.cpp file:



```
PYBIND11_MODULE(FUSINTER_v3_pybind, m) {
    pybind11::class_<lib::FUSINTERDiscretizer>(m, "FUSINTERDiscretizer")
            .def(pybind11::init<float, float>())
            .def("fit", &lib::FUSINTERDiscretizer::fit)
            .def("transform",&lib::FUSINTERDiscretizer::transform);
}
```

- This tells pybind that we want to create a module named FUSINTER_v3_pybind that has a class called FUSINTERDiscretizer.

How to make the C++ code callable

- To compile this code cmake is used.
- First we tell cmake where to find the pybind submodule:

```
add_subdirectory(external/pybind11)
```

- Then we need to tell it to use the activated virtualenv to compile the shared library.

```
set(Python_VIRTUALENV FIRST)
```

- Finally we need to set a target with a provided pybind11 utility function

```
pybind11_add_module(FUSINTER_v3_pybind pybind11.cpp)
```

How to make the C++ code callable

- After creating a build directory we can simply call cmake to populate it. To get the best results we need to set the build type to “release”:

  ```
  cmake -DCMAKE_BUILD_TYPE=Release -S. -B build
  ```

- After calling “make” in the build directory the `.so` file is compiled.
- Now it is possible to open a Python instance and import the module with:

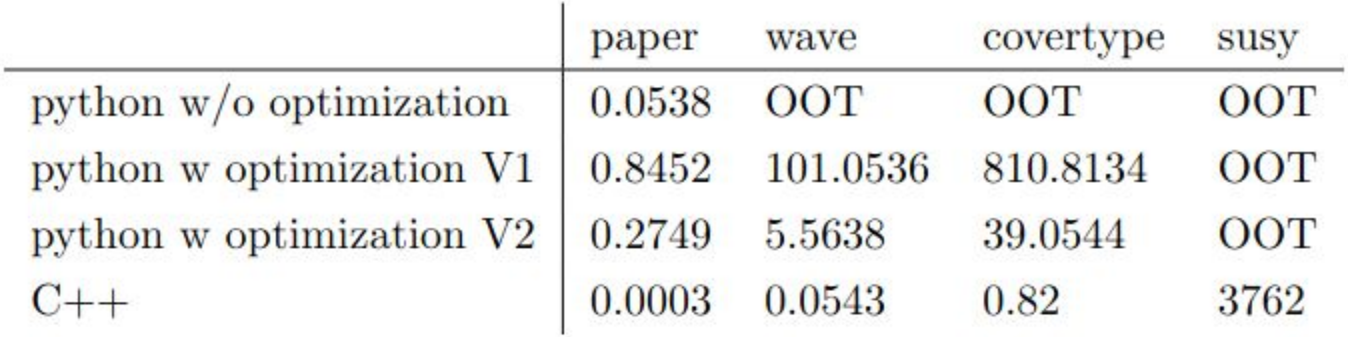
  ```
  import FUSINTER_v3_pybind

  discretizer = FUSINTER_v3_pybind.FUSINTERDiscretizer()
  ```

Benchmarks

Benchmarks

- OOT occurs if the run time was longer than 2 hours.
- The algorithm was applied to all non categorical features of the first 3 datasets and on the first feature of SUSY.
- This results in 90 examples for 1 feature for paper; 5000 examples with 21 features for wave; 581012 examples with 10 features for covertype; 5000000 examples with 1 feature for susy



	paper	wave	covertype	susy
python w/o optimization	0.0538	OOT	OOT	OOT
python w optimization V1	0.8452	101.0536	810.8134	OOT
python w optimization V2	0.2749	5.5638	39.0544	OOT
C++	0.0003	0.0543	0.82	3762

Further Optimizations

Further Optimizations

- Without altering results:

  The algorithm has to apply many delete operation (splits, table after merge,...). I could try to come up with a data structure that deletes efficiently and still allows the needed indexing operations. E.g. Masking instead deleting and then doing bulk deletes after some time.

- With altering results:

  I think that preprocessing datasets before giving them to the algorithm could improve the performance without altering the results too much.

  E.g. Reduce the jitter by fixing the examples to a grid; Using a kernel density estimator and sample a dataset with less examples from the resulting PDF;...