Classical methods and models in MLE with Scikit Learn

BECM33MLE — Machine Learning Engineering

Dr. Tomáš Báča

Multi-Robot Systems group, Faculty of Electrical Engineering
Czech Technical University in Prague

Python Virtual Environment

- always recommend to work in a **Virtual Environment**
- it stores copies of the particular dependencies locally
- one solution to a **dependency hell**

Dependencies (requirements.txt)

```
numpy==2.3.3
matplotlib==3.10.6
scikit-learn==1.7.2
pandas==2.3.2
jupyter
jupyterlab-vim
```

- the version is fixed for some of the packages to make the code within the examples work in the future

Setting up python environment (Ubuntu 24.04)

```
# install the python3's virutal environment
sudo apt get install python3-venv

# create the virtual environmetn
python3 -m venv python-env

# activate the environment
source ./python-env/bin/activate

# install dependencies manually
pip install numpy ...
```

or install dependencies in bulk

```
# install deps from requirements.txt
python3 -m pip install -r requirements.txt
```

Jupyter Notebook & Jupyter Lab

- web-interface with an editor
- allows execution of portions of your code
- remembers the state of variables
- remembers where you left it
- embedded visualization
- well-suited for learning and prototyping
- (bonus) vim-like key bindings
- **Jupyter Notebook**
  - single document, simple
- **Jupyter Lab**
  - multiple documents at once
  - similar to an IDE
  - better file management

Running Jupyter Notebook

```
1 # 1. activate your python env
2 source ./python-env/bin/activate
3
4 # 2. run
5 jupyter notebook
```

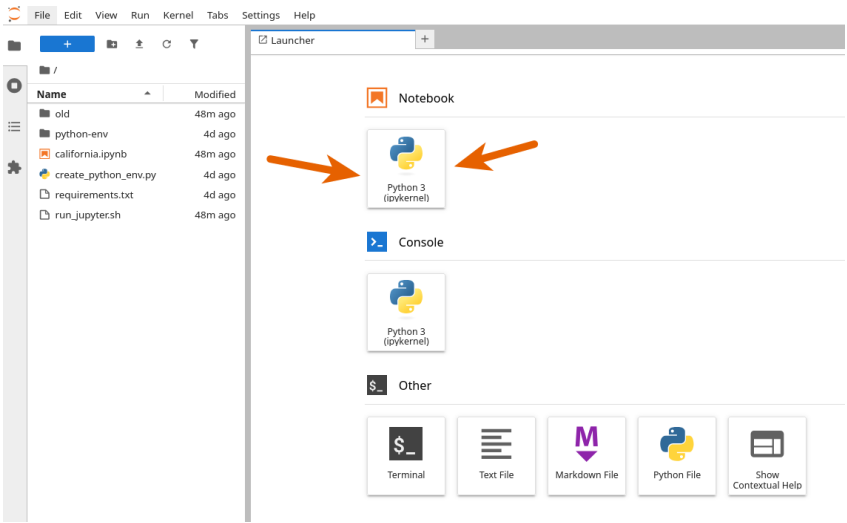
3. check your browser (`localhost:8888`)

Running Jupyter Lab

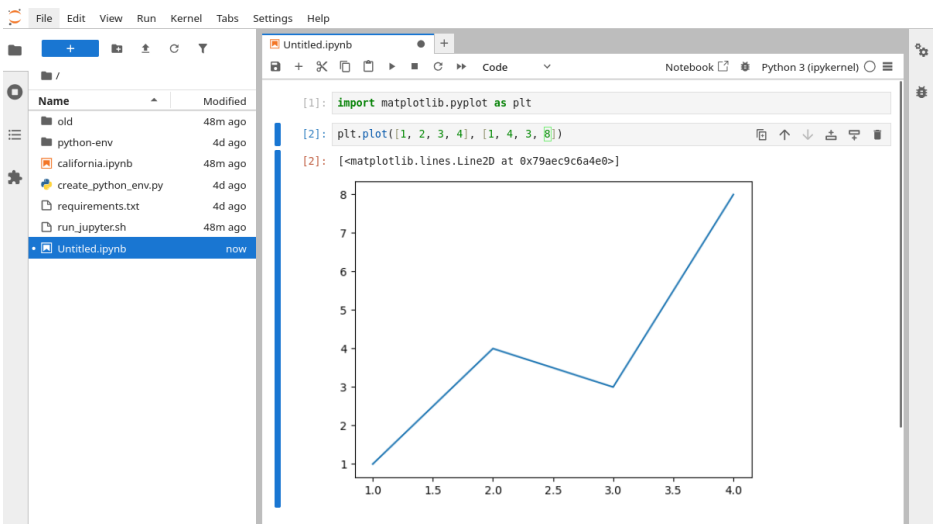
```
1 # 1. activate your python env
2 source ./python-env/bin/activate
3
4 # 2. run
5 jupyter-lab
```

3. check your browser (`localhost:8888`)

Jupyter Notebook & Jupyter Lab



Jupyter Notebook & Jupyter Lab



- **Python library**
- manipulation of tabular data
- *"Excel in Python"*
- can load common formats (CSV, Excel, SQL database, JSON, ...)
- suitable for:
  - data cleaning and processing
  - merging and joining of datasets
  - visualization (Matplotlib integration)
  - handles temporal data
  - integrates with *Numpy* and *SkLearn*

Pandas dataframe

- the main workhorse of Pandas
- 2D, size-mutable data structure

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24

20640 rows × 8 columns

Scikit Learn

- free and open source **Python library**
- implements many classical ML methods suitable for work with *small data*
- perfect for introduction into practical ML
- elegant interface that leads to clean and simple code
- plenty of examples and community support
- `https://scikit-learn.org/stable/user_guide.html`

Scikit learn can do

- Classification
  - NN, SVM, random forests, logistic regression, ...
- Regression
  - gradient boosting, random forests, logistic regression, ...
- Clustering
  - k-Means, hierarchical clustering, ...
- Dimensionality reduction
  - PCA, linear & nonlinear manifold learning
- Model selection, hyper parameter optimization
  - grid search, cross validation, metrics, ...
- Data preprocessing
  - data preprocessing, feature extraction, ...

Importing a toy dataset — California housing

- data matrix:
  - median house age
  - median income
  - avg number of rooms
  - avg number of bedrooms
  - population in census block
  - occupancy
  - latitude
  - longitude
- target value:
  - \$ $\frac{\text{house price}}{100000}$ USD

Importing a dataset

```
1 from sklearn.datasets import
      fetch_california_housing
2 from pandas import pd
3
4 housing = fetch_california_housing(as_frame=True)
```

What is in the dataset?

```
1 # plaintext print
2 print(housing.data)
3 print(housing.target)
4
5 # print in jupyter notebook
6 housing.data
7 housing.target
```

California housing dataset

Data matrix

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24

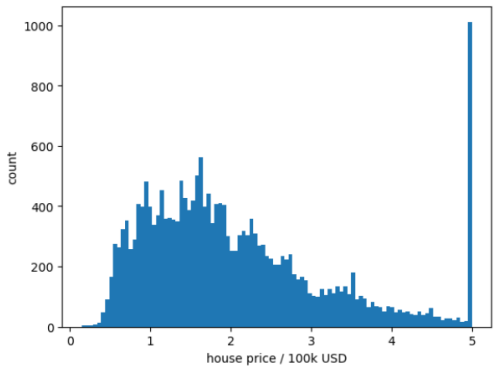
20640 rows × 8 columns

Target value

```
0        4.526
1        3.585
2        3.521
3        3.413
4        3.422
         ...
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
```

California housing dataset — targets

histogram of the house price



Plotting the histogram

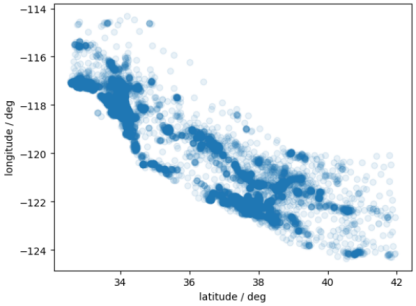
```
1 plt.hist(housing.target, bins=100)
2 plt.xlabel("house price / 100k USD")
3 plt.ylabel("count")
```

Observation?

- note that the maximum price is 5k USD
- the histogram has a clear spike in that price

California housing dataset — locations

Location density of data samples



Plotting the histogram

```
1  plt.scatter(housing.data["Latitude"],
       housing.data["Longitude"], alpha=0.1)
2  plt.xlabel("latitude / deg")
3  plt.ylabel("longitude / deg")
```

California housing dataset — loading for learning

Loading `X` and `y` of the dataset

`return_X_y=True`

```
1 from sklearn.datasets import fetch_california_housing
2
3 X, y = fetch_california_housing(return_X_y=True)
```

X

```
[[   8.3252      41.           6.98412698 ...    2.55555556
    37.88      -122.23      ]
 [   8.3014      21.           6.23813708 ...    2.10984183
    37.86      -122.22      ]
 [   7.2574      52.           8.28813559 ...    2.80225989
    37.85      -122.24      ]
 ...
 [   1.7         17.           5.20554273 ...    2.3256351
    39.43      -121.22      ]
 [   1.8672      18.           5.32951289 ...    2.12320917
    39.43      -121.32      ]
 [   2.3886      16.           5.25471698 ...    2.61698113
    39.37      -121.24      ]]
```

y

```
[4.526 3.585 3.521 ... 0.923 0.847 0.894]
```

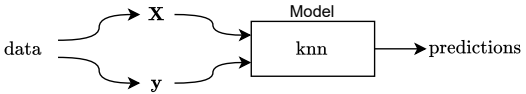
The *simplest* regression model

- the minimal example that *does something*
- using KNN to predict the price by finding nearest neighbors in the dataset and averaging their price

KNN regression pipeline

```
1  from sklearn.datasets import fetch_california_housing
2  from sklearn.neighbors import KNeighborsRegressor
3
4  X, y = fetch_california_housing(return_X_y=True)
5
6  model = KNeighborsRegressor()
7
8  # training
9  model.fit(X, y)
10
11 # making predictions
12 prediction = model.predict(X)
```

"The ML model"



The *simplest* regression model

- the minimal example that *does something*
- using KNN to predict the price by finding nearest neighbors in the dataset and averaging their price

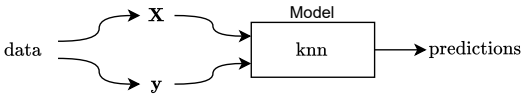
Possible flaws?

- we are using all the data to train the model at once
- we are doing our predictions on the same data we used for training

KNN regression pipeline

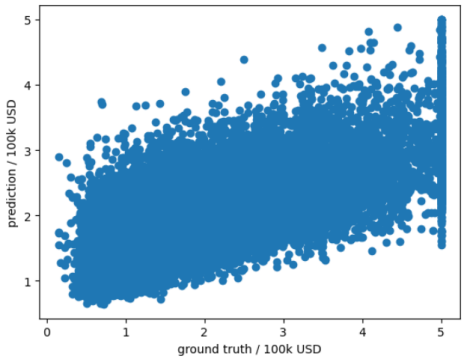
```
1  from sklearn.datasets import fetch_california_housing
2  from sklearn.neighbors import KNeighborsRegressor
3
4  X, y = fetch_california_housing(return_X_y=True)
5
6  model = KNeighborsRegressor()
7
8  # training
9  model.fit(X, y)
10
11 # making predictions
12 prediction = model.predict(X)
```

"The ML model"



The *simplest* regression model — results

Predictions vs. ground truth



Showing the predictions

```
1 plt.scatter(y, pred)
2 plt.xlabel("ground truth / 100k USD")
3 plt.ylabel("prediction / 100k USD")
```

Observations

- resemblance of a predictive behavior is there
- ideally, the data would form a line with slope $= 1$
- see the pattern caused by the capped house price at 500k USD?
- far from ideal, but this is a good start

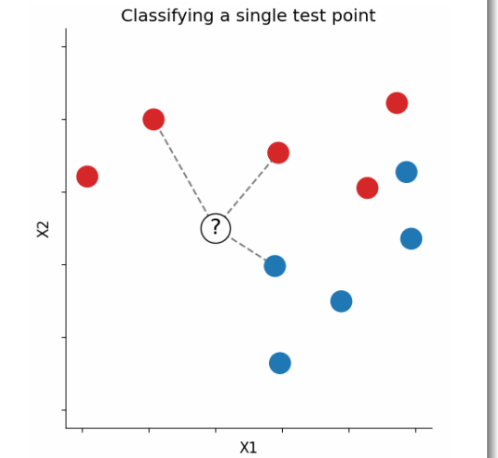
The *simplest* regression model — improving?

- our data has varying physical quantities in different dimensions
- one axis is in feet^2, other in degrees, other in multiples of 100k USD

Solution?

- scaling data such that their distribution is *similar*
- many ways how to do that, but let's start with a simple practical example

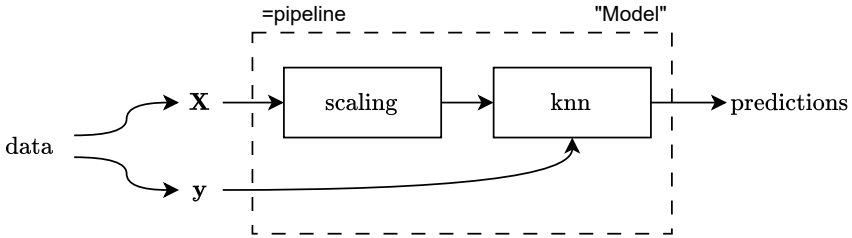
KNN



The *simplest* regression model — creating a pipeline

- let's add a scaling block in front of our KNN regressor
- with two block, we are already forming a *pipeline*

The *simplest* pipeline diagram



Adding preprocessing and pipeline

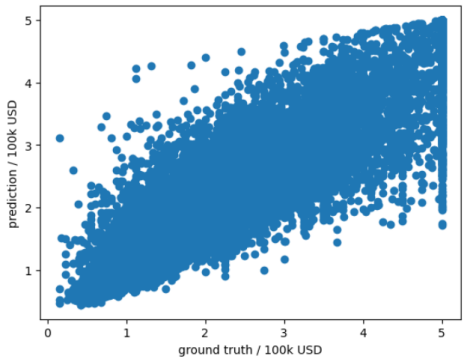
- the `Pipeline` has the same interface as the KNN regressor
  - `.fit(X, y)`
  - `.predict(X)`

KNN regression pipeline

```python
from sklearn.datasets import fetch_california_housing
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

X, y = fetch_california_housing(return_X_y=True)

model = Pipeline([
  ("scaler", StandardScaler()),
  ("predictor", KNeighborsRegressor()),
])

# training
model.fit(X, y)

# making predictions
prediction = model.predict(X)
```

The *simplest* regression model — results

Predictions vs. ground truth now



Showing the predictions

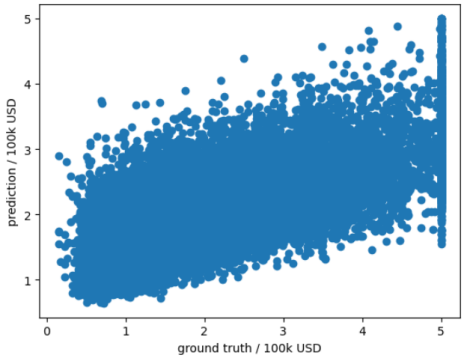
```
1 plt.scatter(y, pred)
2 plt.xlabel("ground truth / 100k USD")
3 plt.ylabel("prediction / 100k USD")
```

Observations

- better performance than before
- as simple to use as before

The *simplest* regression model — results

Predictions vs. ground truth before



Showing the predictions

```
1 plt.scatter(y, pred)
2 plt.xlabel("ground truth / 100k USD")
3 plt.ylabel("prediction / 100k USD")
```

Observations

- better performance than before
- as simple to use as before

1. model parameters

- our model might have parameters that might need changing to improve "our *performance*"

Let's add `n_neighbors=1`

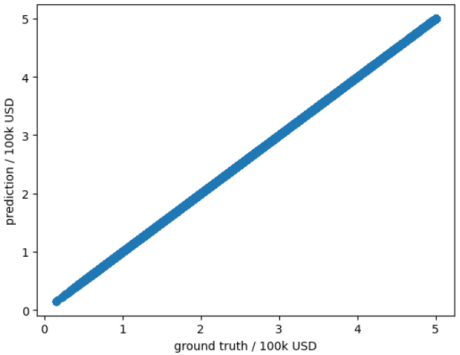
```
...

model = Pipeline([
  ("scaler", StandardScaler()),
  ("predictor", KNeighborsRegressor(
      n_neighbors=1)),
])

...
```

Problems with current approach?

1. model parameters

- our model might have parameters that might need changing to improve "our *performance*"

Predictions vs. ground truth for 1 neighbor



Let's add `n_neighbors=1`

```
...

model = Pipeline([
  ("scaler", StandardScaler()),
  ("predictor", KNeighborsRegressor(
       n_neighbors=1)),
])

...
```

This leads to problem #2

- our performance evaluation method in not very good

Problems with current approach?

2. our evaluation method is wrong

- we are checking performance of the model using the same data we trained on
- this makes selecting the right parameters impossible (for other data than the one we are training on)

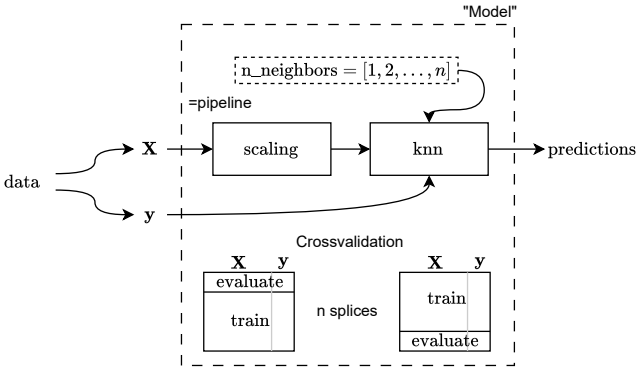
Grid search / hyperparameter optimization

- selecting parameter of *block* that can not be trained by the block itself
- iterates over all combinations of parameters

Crossvalidation

- splicing data to several pieces
- training on majority of the data while testing on the rest
- then swapping the splice

Using *grid search* with *crossvalidation*



Observations

- `GridSearchCV` has the same interface
  - `.fit(X, y)`
  - `.predict(X)`
- whole parts of the pipeline can be disabled/enabled

Checking the results

- in Jupyter notebook

```
1  pd.DataFrame(model.cv_results_)
```

Introducing GridSearchCV

```
1  ...
2
3  from sklearn.model_selection import
       GridSearchCV
4
5  pipeline = Pipeline([
6    ("scaler", StandardScaler()),
7    ("predictor", KNeighborsRegressor()),
8  ])
9
10 model = GridSearchCV(
11   pipeline,
12   param_grid={'predictor__n_neighbors':
13                [1, 2, 3, 4, 5, 6]
14               },
15   cv=3
16 )
17
18 model.fit(X, y)
19
20   ...
```

Grid search results

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_predictor__n_neighbors	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
0	0.009950	0.000354	0.218222	0.016866	1	{'predictor__n_neighbors': 1}	0.324068	0.334830	0.323371	0.327423	0.005245	6
1	0.009837	0.000220	0.246559	0.021540	2	{'predictor__n_neighbors': 2}	0.468788	0.503457	0.424388	0.465544	0.032361	5
2	0.009659	0.000020	0.269420	0.019367	3	{'predictor__n_neighbors': 3}	0.518547	0.543340	0.473595	0.511827	0.028867	4
3	0.009642	0.000061	0.275874	0.021485	4	{'predictor__n_neighbors': 4}	0.540323	0.564974	0.499827	0.535041	0.026857	3
4	0.010023	0.000212	0.289105	0.020307	5	{'predictor__n_neighbors': 5}	0.551149	0.579313	0.511781	0.547414	0.027696	2
5	0.009678	0.000049	0.305267	0.028973	6	{'predictor__n_neighbors': 6}	0.558435	0.586185	0.521134	0.555251	0.026652	1

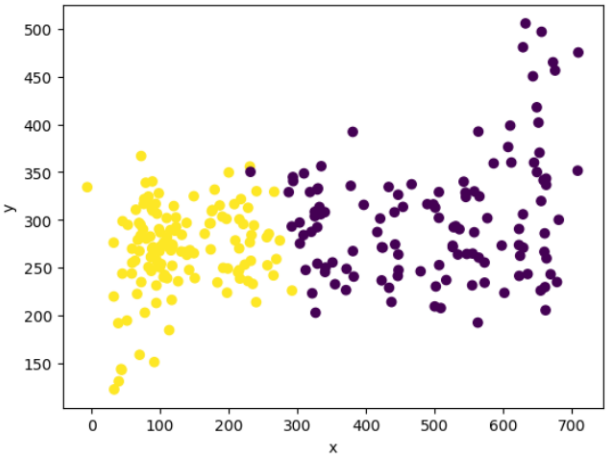
Observations

- the best KNN model is the one with 6 neighbors (makes sense)
- the model won with *a mean_test_score* of 0.555
  - what score is it
  - how is it calculated?
  - how do we change it?
  - we will get back to this

Observations

- two classes
- should be nicely separable
- outliers near the edges
- data in arbitrary range on both axes

A dataset with outliers



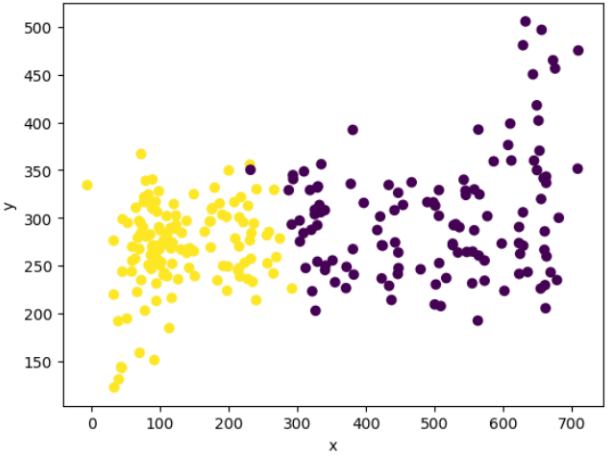
Observations

- two classes
- should be nicely separable
- outliers near the edges
- data in arbitrary range on both axes

Let's try standard scaler

- what does it even do?
- how will it perform?
- will it be influenced by the outliers?

A dataset with outliers



Standard scaler

$$z = \frac{x - \mu}{\sigma},$$

where

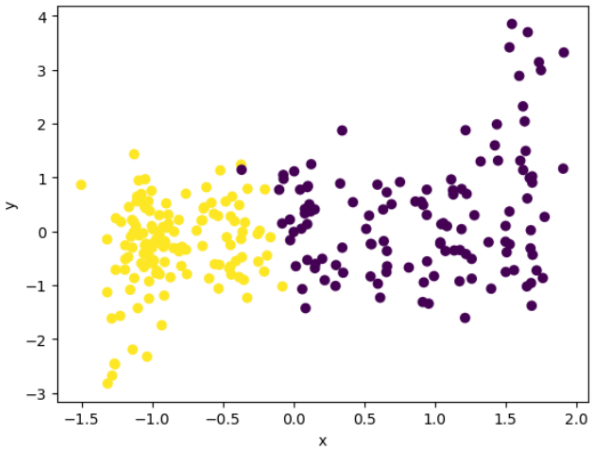
$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}$$

Observations

- outliers still visible
- axes' range is *near zero*
- the shape of the data is similar

Dataset after transformation



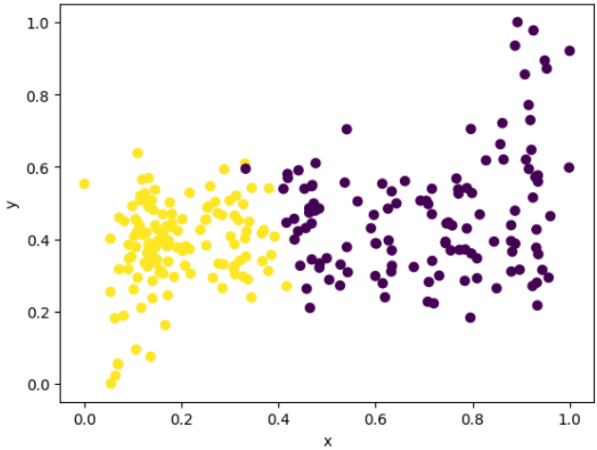
MinMax scaler

$$z = \frac{x - \min}{\max - \min},$$

Observations

- outliers still visible
- axes' range is between 0 and 1
- distribution is preserved

Dataset after transformation



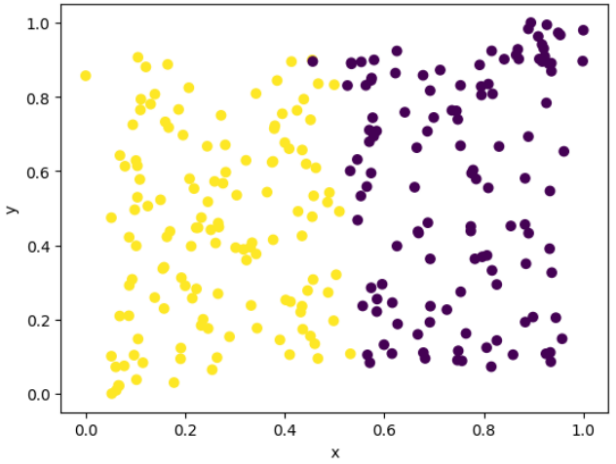
Quantile transformer

- reshapes the data to have uniform (or normal) distribution
- robust scaler

Observations

- uniform output (default)
- outliers are not prominent anymore
- non-linear transformation
  - will break linear correlations between features

Dataset after transformation (uniform, 10 quantiles)



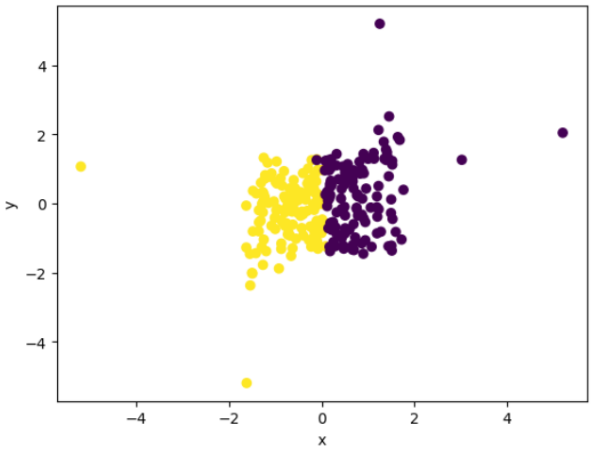
Quantile transformer

- reshapes the data to have uniform (or normal) distribution
- robust scaler

Observations

- uniform output (default)
- outliers are not prominent anymore
- non-linear transformation
  - will break linear correlations between features

Dataset after transformation (normal, 10 quantiles)



Linear separability

Problem?

- classes are not linearly separable
- a.k.a., a single separating hyperplane can not be found

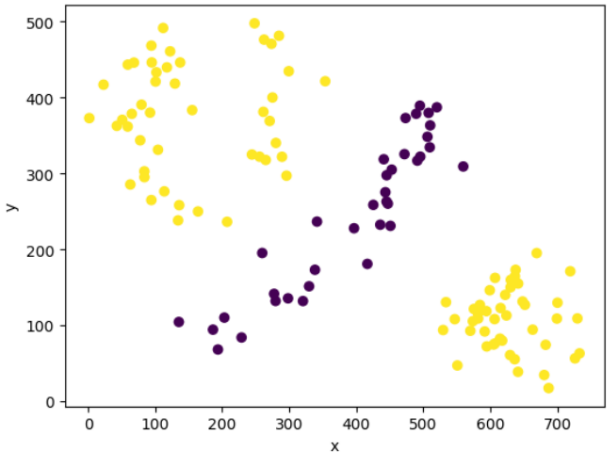
Solution?

- use more complex classifier
  - nonlinear
  - e.g., boosting
  - decision tree

Easier solution?

- use more complex classifier
- lift feature to new dimension

Linearly non-separable two-class dataset



Problem?

- classes are not linearly separable
- a.k.a., a single separating hyperplane can not be found

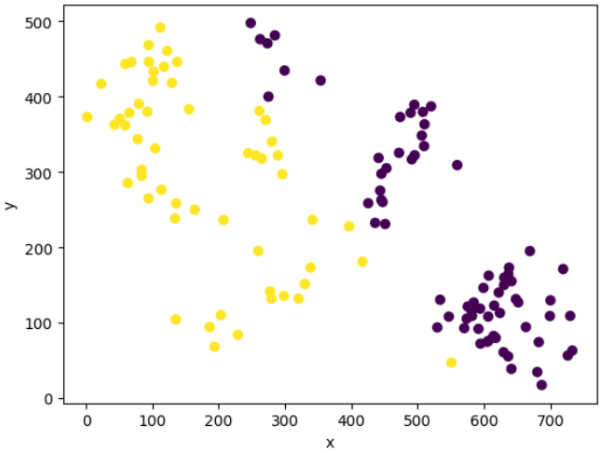
Solution?

- use more complex classifier
  - nonlinear
  - e.g., boosting
  - decision tree

Easier solution?

- use more complex classifier
- lift feature to new dimension

Badly-classified by linear classifier



Polynomial features

Lifting features using polynomial interactions

- example for order $= 2$

$$\mathbf{x} = \left[x_1, x_2\right]^\intercal$$

$$\mathbf{x}_{\text{new}} = \Phi(\mathbf{x}) = \left[1, x_1, x_2, x_1^2, x_1 x_2, x_2^2\right]^\intercal$$

Observations

- leads to simple classifier, which can be robust
- e.g., all favourite SVM
- makes computations more expensive during production
- can be simplified with *the kernel trick*

Using polynomial features

```
1   ...
2
3   from sklearn.preprocessing import
        PolynomialFeatures
4   from sklearn.linear_model import
        LogisticRegression
5
6   pipeline = Pipeline([
7     ("scale", PolynomialFeatures()),
8     ("classifier", LogisticRegression(
          max_iter=1000, class_weight="balanced
          ")),
9   ])
10
11  pred = pipeline.fit(X, y).predict(X)
12
13    ...
```

Polynomial features

Observations

- simple to use
- just like any other Sklearn transformer

Logistic regression with polynomial features

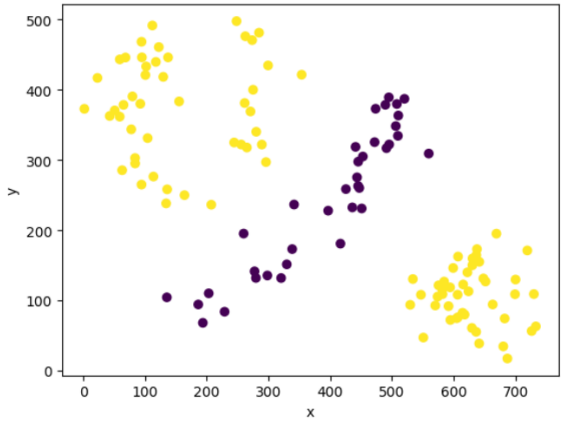
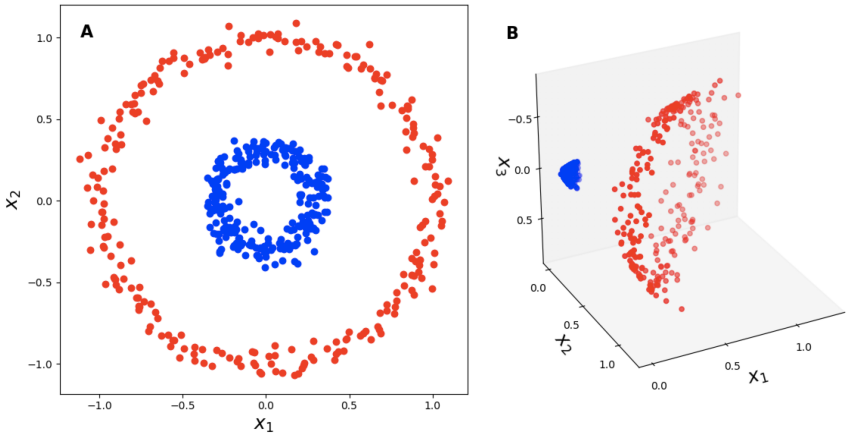


Illustration of lifting from 2D to 3D



Source: `https://gregorygundersen.com/blog/2019/12/10/kernel-trick/`

Encoding text and categorical attributes

How to encode text?

- added uunique ID in 1D space might cause problems
- better way?
  - give each class a unique unit vector in its own dimension
  - this this creates as many dimensions (+1) as the # of classes

Even better way?

- learn unique vectors in lower dimensional space
- called **embedding**
- heart (mouth) of transformers

Using OneHot encoder

```
from sklearn.preprocessing import OneHotEncoder

X = [['red'], ['green'], ['blue'], ['red']]

encoder = OneHotEncoder(handle_unknown="ignore")
encoder.fit(X)

new_X = encoder.transform(X).todense()
```

```
matrix([[0., 0., 1.],
        [0., 1., 0.],
        [1., 0., 0.],
        [0., 0., 1.]])
```

Inverse:

```
encoder.inverse_transform([[0, 1, 0]])
```

```
[['green']]
```

Objects in Scikit

Estimator

- implements `.fit()`
- only consumes data
- e.g., some statistics collector

Transformer

- implements `.fit()`, `.transform()`
- scalers
- dimensionality reducers (PCA)
- imputers (handle missing values)

Predictor

- implements `.fit()`, `.predict()`
- regressors, classifiers

Pipeline

- is a Predictor

GridSearchCV

- is a Predictor

Binary classification metrics — Precision, Accuracy and Recall

Precision

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- given I mark the sample as relevant, high often am I right

Recall

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

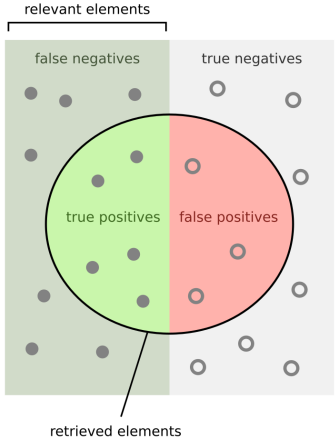
- did I marked all the relevant instances?

Accuracy

$$\text{accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- how precise am I in marking correctly both instances

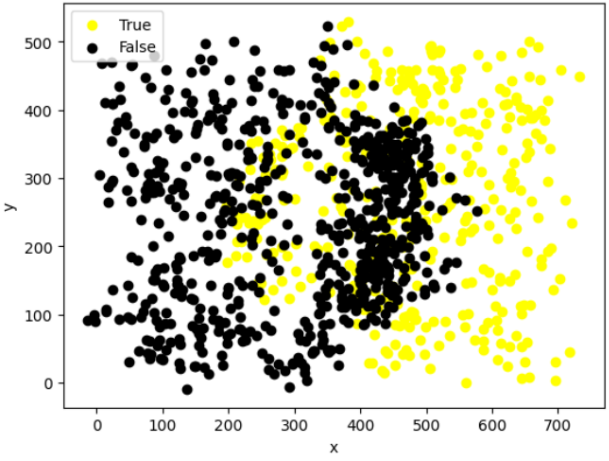
Diagram of possible results



Is there a single best classifier?

- no, it depends on our metrics
- do we care more about
  - precision?
  - accuracy?
  - recall?
- or some combination of these?

Two non-separable classes



Logistic regression with default metrics

Observations

- we are trying Logistic Regression
- we are varying the class weight
- we are searching for the *best-performing* predictor

Let's run this

- let's run it as it is
- the default score for logistic regression is *Accuracy*

Trying logistic regression

```
...

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
  ("classifier", LogisticRegression(max_iter=1000,
       verbose=0)),
])

model = GridSearchCV(
  estimator=pipeline, cv=4, param_grid={
  'classifier__class_weight': [{0: 5, 1: v} for v in
       range(1, 10)]},
)

pred = model.fit(X, y).predict(X)

...
```

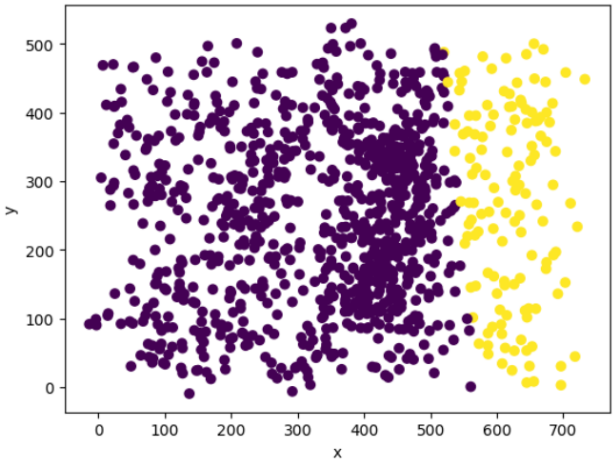
Observations

- the decision is *cut* at a reasonable compromise
- many FP as well as FN

Metrics on the dataset

- **accuracy** = 0.77
- precision = 0.97
- recall = 0.35

Logistic Regression selected by the Accuracy score



Logistic regression with Precision metrics

Observations

- we are trying Logistic Regression
- we are varying the class weight
- we are searching for the *best-performing* predictor

Scoring

- new `scoring` argument
  - we can add arbitrary scoring functions
  - all will be evaluated during the grid search
- new `refit` argument
  - the pipeline will be re-trained using parameters that won using the pre-defined score

Logistic regression precision

```
1  from sklearn.linear_model import LogisticRegression
2  from sklearn.model_selection import GridSearchCV
3  from sklearn.metrics import precision_score,
       accuracy_score, recall_score
4
5  pipeline = Pipeline([
6    ("classifier", LogisticRegression(max_iter=1000,
         verbose=0)),
7  ])
8
9  model = GridSearchCV(
10   estimator=pipeline, cv=4, param_grid={
11   'classifier__class_weight': [{0: 5, 1: v} for v in
         range(1, 10)]},
12   scoring={'precision': make_scorer(precision_score)
         , 'recall': make_scorer(recall_score), '
         accuracy': make_scorer(accuracy_score)},
13   refit='precision',
14 )
15
16 pred = model.fit(X, y).predict(X)
```

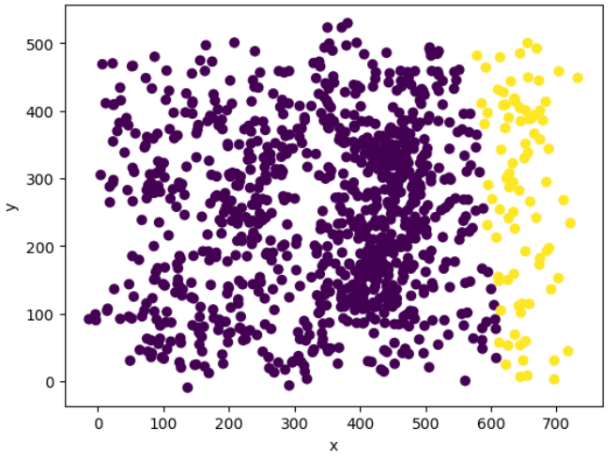
Observations

- the decision is *cut* at the very right
- minimizes false positives
- lot of missed *true* sample

Metrics on the dataset

- accuracy = 0.73
- **precision** = 1.0
- recall = 0.23

Logistic Regression selected by the Precision score



Logistic regression with Recall metrics

Observations

- we are trying Logistic Regression
- we are varying the class weight
- we are searching for the *best-performing* predictor

Scoring

- new `scoring` argument
  - we can add arbitrary scoring functions
  - all will be evaluated during the grid search
- new `refit` argument
  - the pipeline will be re-trained using parameters that won using the pre-defined score

Logistic regression recall

```
1  from sklearn.linear_model import LogisticRegression
2  from sklearn.model_selection import GridSearchCV
3  from sklearn.metrics import precision_score,
       accuracy_score, recall_score
4
5  pipeline = Pipeline([
6    ("classifier", LogisticRegression(max_iter=1000,
         verbose=0)),
7  ])
8
9  model = GridSearchCV(
10   estimator=pipeline, cv=4, param_grid={
11   'classifier__class_weight': [{0: 5, 1: v} for v in
         range(1, 10)]},
12   scoring={'precision': make_scorer(precision_score)
         , 'recall': make_scorer(recall_score), '
         accuracy': make_scorer(accuracy_score)},
13   refit='recall',
14 )
15
16 pred = model.fit(X, y).predict(X)
```

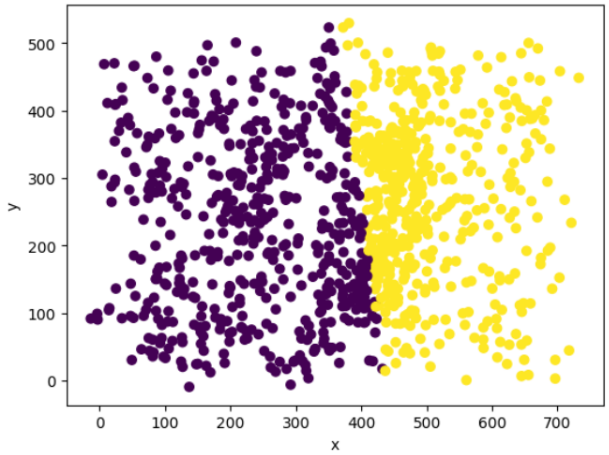
Observations

- the decision is *cut* at the very right
- minimizes false positives
- lot of missed *true* sample

Metrics on the dataset

- accuracy = 0.67
- precision = 0.52
- **recall** = 0.72

Logistic Regression selected by the Precision score



Storing a trained sklearn model

- as simple as dumping the object into a file
- alternatively, you can use pickle

Storing a model

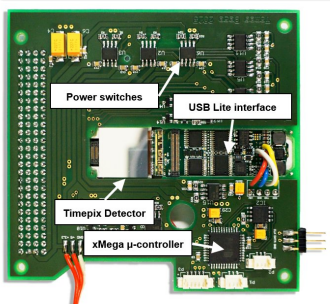
```
1 ...
2
3 from joblib import dump
4
5 dump(model, "my_model.joblib")
6
7 ...
```

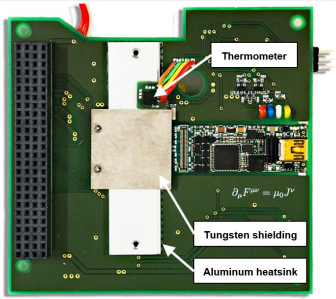
Loading a model

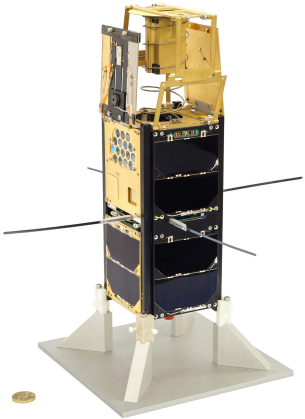
```
1 ...
2
3 from joblib import load
4
5 model = load("my_model.joblib")
6
7 ...
```

VZLUSAT-1 — The first Czech CubeSat [1], [2]

- Custom-designed embedded Timepix board and software [2]
- Onboard image processing, filtering, automated acquisition
- The longest-operating Czech(-oslovakian) sat. (2017–2023)





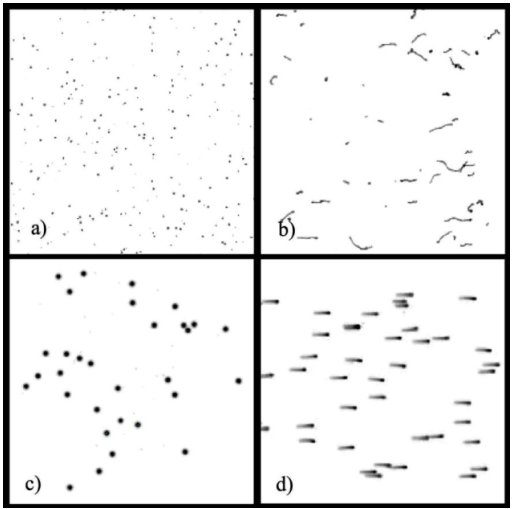


Timepix - Ionizing radiation dosimetry and imaging

- real-time recording of the interaction of an incoming particle and the sensor
- machine learning was applied to distinguish particle types [3]
- no dark-current noise (the images are spotless besides the actual data)

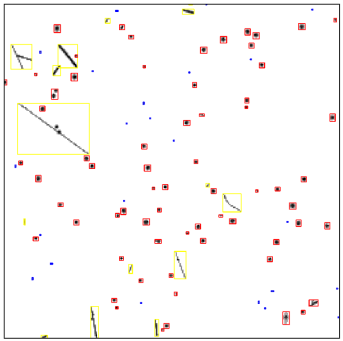
Examples in the Figure

- photons (gamma)
- electrons (beta)
- Helium nuclei (alpha)
- protons



Random forests for particle track classification

Particle classification in a single image [4]

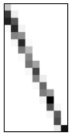


(a) Image from the Timepix detector.

(b) dot

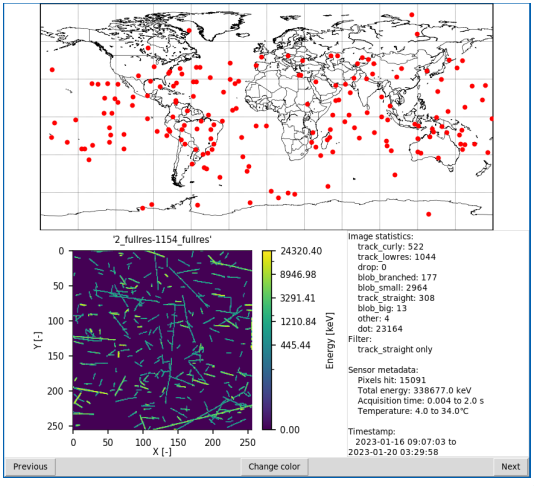
(c) blob

(d) drop

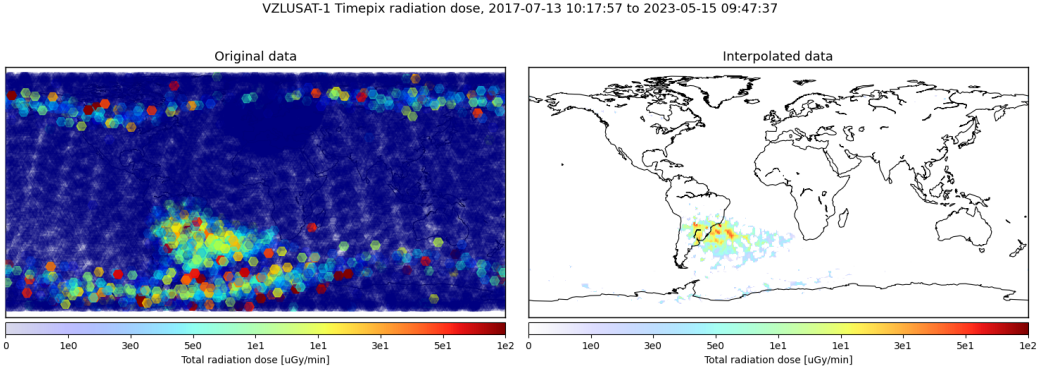


(e) track

Image browser



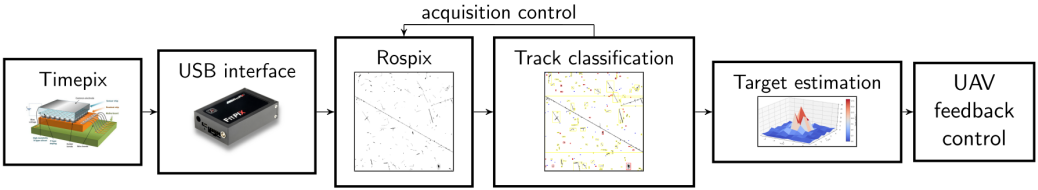
Plot for Beta particles



https://github.com/vzlusat/vzlusat1-timepix-data [4], [5]

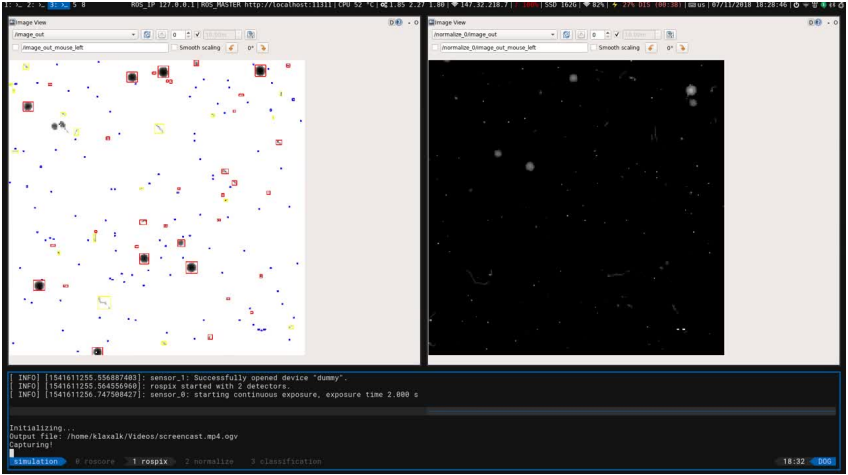
Proof of concept for UAVs

- Real-time particle track classifier for mobile robots and drones.
- We have finished well-evaluated TACR grant to develop this tech.



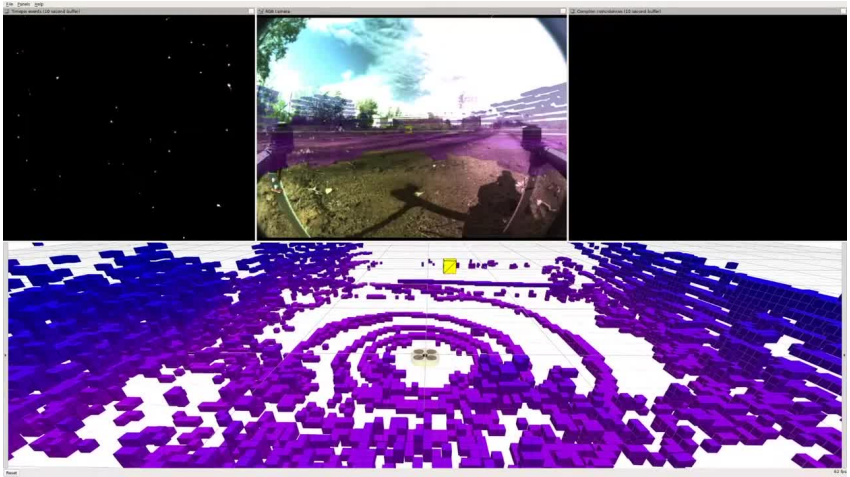
[4] T. Baca, M. Jilek, P. Manek, P. Stibinger, V. Linhart, J. Jakubek, *et al.*, "Timepix Radiation Detector for Autonomous Radiation Localization and Mapping by Micro Unmanned Vehicles," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2019, pp. 1–8

Realtime classification of ionizing particles in ROS



Video: https://youtu.be/8rZrbmDHG4E

Endgame: realtime search for ionizing radiation sources



Video: `https://youtu.be/oH4jMMHfGVA`

Thanks for your attention

References I

[1] M. Urban, O. Nentvich, V. Stehlikova, T. Baca, V. Daniel, and R. Hudec, "VZLUSAT-1: Nanosatellite with miniature lobster eye X-ray telescope and qualification of the radiation shielding composite for space application," *Acta Astronautica*, vol. 140, pp. 96–104, 2017.

[2] T. Baca, M. Platkevic, J. Jakubek, *et al.*, "Miniaturized X-ray telescope for VZLUSAT-1 nanosatellite with Timepix detector," *Journal of Instrumentation*, vol. 11, no. 10, p. C10007, 2016.

[3] T. Baca, M. Jilek, I. Vertat, *et al.*, "Timepix in LEO Orbit onboard the VZLUSAT-1 Nanosatellite: 1-year of Space Radiation Dosimetry Measurements," *Journal of Instrumentation*, vol. 13, no. 11, p. C11010, 2018.

[4] T. Baca, M. Jilek, P. Manek, *et al.*, "Timepix Radiation Detector for Autonomous Radiation Localization and Mapping by Micro Unmanned Vehicles," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2019, pp. 1–8.

[5] T. Baca, P. Stibinger, D. Doubravova, *et al.*, "Gamma Radiation Source Localization for Micro Aerial Vehicles with a Miniature Single-Detector Compton Event Camera," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2021, pp. 1–9.