5th Session

Martin Biehler

Aug 12, 2022

CONTENTS

1 Data Modelling for data with temporal context **3**

2 Overfitting and Cross-Validation **5**

3 Cross-Validation **7**

4 Date Leakage and Dependent Data **9**

5 Sources of data leakage **11**
5.1 train data contains features that are not available in production . . . 11
5.2 future data somehow slipped into the training set . . . 11
5.3 there is one feature that interacts with the target . . . 12
5.4 Some more cases where we have data leakage: . . . 12
5.5 Example: credit card applications . . . 13
5.6 Solution: . . . 17

6 Dependency between data-samples **19**
6.1 Some more cases where we have dependent data . . . 19

7 Creditcard-Fraud: imbalanced data **23**
7.1 the data-set is too large - we subsample the majority class . . . 23

8 what will our accuracy be ad hoc? **25**

9 get a model **27**

10 train / test split **29**

11 proper test set? **35**

12 Strategies for the imbalanced case: **37**
12.1 oversample fraud . . . 37
12.2 weigh minority class . . . 40
12.3 Undersample Fraud . . . 41
12.4 do both: oversample minority class and undersample majority class . . . 42
12.5 get best combination: sampling and hyper-parameters . . . 42
12.6 One-Class SVM . . . 43

13 Titanic - Machine Learning from Disaster **45**
13.1 Predict survival on the Titanic and get familiar with ML basics . . . 45
13.2 continuous variables . . . 45
13.3 categorical or discrete variables . . . 45

13.4 ordinal variables . . . 46

14 Processing of Variable **47**
14.1 Continuous Variables . . . 47
14.2 Categorical or discrete variables . . . 47
14.3 ordinal data . . . 48

15 Missing Data **49**
15.1 Interactions . . . 50
15.2 Standardization / Normalization . . . 50

16 not covered here **53**

17 Knowledge Discovery -> Data Mining -> Data Science -> Machine Learning **55**

18 Lineare Regression **57**

19 multivariate case: more than one x variable **59**

20 Polynomial regression as an example for more than one variable **61**
20.1 Overfitting . . . 62
20.2 perfect fit: as many variables as data samples . . . 63
20.3 Bias-Variance Tradeoff . . . 64

21 Dealing with overfitting **67**
21.1 Ridge regression . . . 67
21.2 Lasso . . . 68

22 Extension: logistic regression and the GLM **71**
22.1 exponential family of distributions . . . 71

23 Neural Network **73**
23.1 classical linear regression . . . 74
23.2 logistic regression . . . 75

24 Why trees? **77**
24.1 Compare to linear Regression (logistic regression in this case) . . . 79
24.2 Splitting criteria . . . 81

25 Genetic Algorithms **83**
25.1 Evolutionary Decision Trees: . . . 84
25.2 my opinion about evolutionary algorithms: . . . 84
25.3 Random Forest . . . 85

26 Gradient Boosted Trees **87**
26.1 Explanation of Gradient Boosted Trees: . . . 87
26.2 most important parameters for stochastic gradient-boosting: . . . 88

27 Clustering **89**
27.1 SOM . . . 90
27.2 Some considerations regarding the scale of used variables . . . 105
27.3 Spectral clustering . . . 107
27.4 Example: 20 newsgroups . . . 111
27.5 Organization . . . 111

28 Tf-idf **113**
28.1 Rand Index 113
28.2 Connectivity / adjacency / affinity matrix 114

29 our choice **115**

30 Approximate nearest neighbors **117**

31 DBSCAN **123**

32 SOM **125**

```
Der Input diese Skripts ist anlässlich eines internen workshops mit Finnova␣
↪entstanden. Im wesentlichen basiert es auf Ausschnitten eines Seminars, das ich␣
↪zuletzt im Sommersemester 2022 an der FHNW gegeben habe.
```

1. Morning
   - **Data Modelling & Cross Validation**
   - data leakage & dependent data
   - imbalanced data (example in python)
   - study: Ebanking Fraud
   - Q&A
2. Afternoon
   - Data Basics & historical perspective
   - Linear Regression
   - Trees
   - house prices (regression example in python)
   - Clustering
   - bonus: Hyperparameter Optimization and AutoML
   - Q&A

CHAPTER

ONE

DATA MODELLING FOR DATA WITH TEMPORAL CONTEXT

CHAPTER

TWO

OVERFITTING AND CROSS-VALIDATION

All classification and regression algorithms are prone to overfitting: The algorithms learn pecularities of the train-data, that are not present in the real-world data. When over-fitted, the algorithms are not generalizing to the real data.

Capacity refers to the ratio of free parameters and the amount of training data.

CHAPTER

THREE

CROSS-VALIDATION

In most real-word applications we do not know the data universe, i.e. we do not know all possible data points that might be there. Our training data is possibly just a biased subsample of the population. When we fit our algorithm to such a subsample its performance will degrade, when applied to new, unseen data points. In order to have an idea, how well our algorithm will perform in such cases, we can use a cross-validation scheme: In the example below, a 5-fold cross-validation is illustrated.

- split the training data in 5 equal sized parts. In *sklearn* you can choose *StratifiedKFold*, that essentially tries to keep the percentages of all classes stable within each fold.
- train your algorithmm on 4 folds and classify data in the 5th hold-out fold. Keep the performance on this fold.
- repeat the last step 4 more times and use each time another fold as your hold-out fold.
- at the end, you have 5 independent estimates of your algorithm's performance
- compute the mean of theses 5 estimates for an overall estimate

1. Morning
   - Data Modelling & Cross Validation
   - **data leakage & dependent data**
   - imbalanced data (example in python)
   - study: Ebanking Fraud
   - Q&A
2. Afternoon
   - Data Basics & historical perspective
   - Linear Regression
   - Trees
   - house prices (regression example in python)
   - Clustering
   - bonus: Hyperparameter Optimization and AutoML
   - Q&A

CHAPTER

FOUR

DATE LEAKAGE AND DEPENDENT DATA

CHAPTER

FIVE

SOURCES OF DATA LEAKAGE

5.1 train data contains features that are not available in production

e.g., the row-number contains information about the target: first come the negative examples, the positive cases were then simply inserted underneath.

5.2 future data somehow slipped into the training set

e.g. Giba's property: taken from kaggle

Giba

1st place

The Data "Property"

Posted in santander-value-prediction-challenge 3 years ago

293

Hello everybody,
Since organizers officially declared that exploring the data property is good enough for the top solutions I decided to share my finding with the community to make that game fair to all participants while is time.

The dataset IS a time series in both dimensions, row wise and column wise :-)
And the column "f190486d6" is the last time stamp present in the dataset. So the target is 2 steps in the future of "f190486d6".

Look that example: kernel

This is one of the property I found. But I believe it can contain more.

Giba

Quote | Follow | Report | 293 Upvoters

and here is the mentioned data-structure: this kernel exploits the leakage

ID	target	f190486d6	58e2e02e6	eeb9cd3aa	9fd594eec	6eef030c1	15ace8c9f	fb0f5dbfe	58e05
7862786dc	3513333.3	0	1477600	1586889	75000	3147200	466461.5	1600000.0	0.0
c95732596	160000.0	310000	0	1477600	1586889	75000	3147200.0	466461.5	16000
16a02e67a	2352551.7	3513333	310000	0	1477600	1586889	75000.0	3147200.0	46646
ad960f947	280000.0	160000	3513333	310000	0	1477600	1586888.9	75000.0	31472
8adafbb52	5450500.0	2352552	160000	3513333	310000	0	1477600.0	1586888.9	75000
fd0c7cfc2	1359000.0	280000	2352552	160000	3513333	310000	0.0	1477600.0	15868
a36b78ff7	60000.0	5450500	280000	2352552	160000	3513333	310000.0	0.0	14776
e42aae1b8	12000000.0	1359000	5450500	280000	2352552	160000	3513333.3	310000.0	0.0
0b132f2c6	500000.0	60000	1359000	5450500	280000	2352552	160000.0	3513333.3	31000
448efbb28	1878571.4	12000000	60000	1359000	5450500	280000	2352551.7	160000.0	35133
ca98b17ca	814800.0	500000	12000000	60000	1359000	5450500	280000.0	2352551.7	16000
2e57ec99f	307000.0	1878571	500000	12000000	60000	1359000	5450500.0	280000.0	23525
fef33cb02	528666.7	814800	1878571	500000	12000000	60000	1359000.0	5450500.0	28000

5.3 there is one feature that interacts with the target

taken from Breast Cancer Identification: KDD CUP Winner's Report Distribution of malignant (black) and benign (gray) candidates depending on patient ID on the X-axis in log scale.

5.4 Some more cases where we have data leakage:

- Customer advisor has a long call with customer and finally sells the product that is shipped only two weeks later. Variables 'last advisory contact' and 'length of call' certainly anticipate the product sale. When an algorithm learns to predict product propensity based on 'last advisor contact' it will ultimately suggest customers to the advisors for whom the advisor has already closed the deal.
- Train and test data is normalized with common sample statistics belonging to the whole data set
  - target encoding is dangerous: we will talk about it later on

- stacking is dangerous: we will discuss this topic as well

5.5 Example: credit card applications

This data example was used in "Econometric Analysis" (William H. Greene) without the author noticing the problem: example taken from here

- card: Dummy variable, 1 if application for credit card accepted, 0 if not
- reports: Number of major derogatory reports
- age: Age n years plus twelfths of a year
- income: Yearly income (divided by 10,000)
- share: Ratio of monthly credit card expenditure to yearly income
- expenditure: Average monthly credit card expenditure
- owner: 1 if owns their home, 0 if rent
- selfempl: 1 if self employed, 0 if not.
- dependents: 1 + number of dependents
- months: Months living at current address
- majorcards: Number of major credit cards held
- active: Number of active credit accounts

```
import pandas as pd

url = 'https://raw.githubusercontent.com/YoshiKitaguchi/Credit-card-verification-
↪project/master/AER_credit_card_data.csv'
df = pd.read_csv(url, error_bad_lines=False, true_values = ['yes'], false_values = [
↪'no'])
print(df.head())
```

```
   card  reports       age  income     share  expenditure  owner  selfemp  \
0  True        0  37.66667  4.5200  0.033270   124.983300   True    False
1  True        0  33.25000  2.4200  0.005217     9.854167  False    False
2  True        0  33.66667  4.5000  0.004156    15.000000   True    False
3  True        0  30.50000  2.5400  0.065214   137.869200  False    False
4  True        0  32.16667  9.7867  0.067051   546.503300   True    False

   dependents  months  majorcards  active
0           3      54           1      12
1           3      34           1      13
2           4      58           1       5
3           0      25           1       7
4           2      64           1       5
```

```
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
import numpy as np
import lightgbm

y = df['card']
X = df.drop('card', axis=1)
```

```
model = lightgbm.LGBMClassifier(boosting_type='gbdt', num_leaves=31, max_depth=- 1,␣
↪learning_rate=0.1,
                                n_estimators=500, subsample_for_bin=20000, objective=
↪'binary',
                                subsample=1.0, subsample_freq=0, colsample_bytree=1.0,
                                n_jobs=- 1, silent=True, importance_type='split',
                                is_unbalance = False, scale_pos_weight = 1.0)
model_pipe = make_pipeline(model)
cv_scores = cross_val_score(model_pipe, X, y, scoring='accuracy')
print(np.mean(cv_scores))
```

```
0.9765065099665862
```

```
from sklearn.inspection import permutation_importance
from matplotlib import pyplot as plt
model.fit(X, y)
result = permutation_importance(model, X, y,
        n_repeats=30,
        random_state=0)
sorted_idx = result.importances_mean.argsort()

fig, ax = plt.subplots()
ax.boxplot(result.importances[sorted_idx].T,
           vert=False, labels=X.columns[sorted_idx])
ax.set_title("Permutation Importances")
fig.tight_layout()
plt.show()
```

```
display(X.loc[y == True, 'expenditure'].mean(), X.loc[y == False, 'expenditure'].
↪mean())
```

```
238.60242068103616
```

```
0.0
```

```
display(X.loc[y == True, 'share'].mean(), X.loc[y == False, 'share'].mean())
```

```
0.08848152972453567
```

```
0.0004767954841216091
```

```
from sklearn import tree
from dtreeviz.trees import *
import matplotlib.pyplot as plt
classifier = tree.DecisionTreeClassifier(max_depth=3)  # limit depth of tree
classifier.fit(X, y)


viz = dtreeviz(classifier,
               X.values,
               y.values,
```

(continues on next page)

(continued from previous page)

```
                target_name='credit-card application',
                feature_names=X.columns.tolist(),
                class_names = ['not_accepted', 'accepted']
               )

viz.save("decision_tree.svg")
```

```
<IPython.core.display.SVG object>
```

5.6 Solution:

Obviously,

- share: Ratio of monthly credit card expenditure to yearly income
- expenditure: Average monthly credit card expenditure

are features that suppose the applicant was granted a credit card.

CHAPTER

SIX

DEPENDENCY BETWEEN DATA-SAMPLES

Training Machine Learning Algorithms works best, when we have many independent data samples in the training data. Dependent data arises when:

- we take repeatedly measures from the same individual (the trained algorithms will not generalize to other individuals)
- we take samples only from one bank (the socia-demographic structure of GKB's customers might be differentfrom that of BCG's customers - as a result the algorithm will badly generalize)

6.1 Some more cases where we have dependent data

- Repeatedly sampling data from the same individual:
  - Fraud: A fraudster commits many frauds that have a similiar pattern; For example for every fraud commited, the fraudster uses a different account of the same bank in Thailand. When doing cross-validation we have frauds related to the very bank in Thailand in the training set as well as in the test set. Hence, we will overestimate the capability of the trained classifier to generalize to new, unseen fraud cases. But it will be very efficient to detect this one fraudster with bank accounts in Thailand.
  - customer journey: to detect an event as soon as possible, data is sampled with different offsets before the event's occurence. When trying to predict an event we could be tempted to sample data from different points in time before the event. This data will always be very similar and is hence dependent. For example, medical health records are not changing very fast and blood pressure two months ago will be similar to that measured one month ago. Most bank accounts have a similar balance in a one months distance.
  - classifying websites: social media websites belong all to facebook. There are just not enough social media websites to learn something about them in the training set and generalize to other social media websites in the test set.
- Train and test data is normalized with common sample statistics belonging to the whole data set
  - target encoding is dangerous: we will talk about it later on
  - stacking is dangerous: we will discuss this topic as well
- Sentence Classification: sentences belonging to the same document
  - customer churn: An angry customer sends frequent e-mails. All e-mails happen to have the same characteristics, e.g. instead of the *Umlaut* 'ü', the customer uses 'ue'. The algorithm might be tempted to learn that 'ue' is a special churn-characteristic. When half of the e-mails end up in the train set and the rest in the test set, we will overestimate the prediction accuracy of the learned algorithm.
  - When building a classifier to distinct medical publications from IT-related publications, it is important to have a representative sample of medical topics as well as tech-topics. When taking sentences from one document that is heavily Java related, the algorithm will struggle to generalize to the programming language Python.

When the 'Java-sentences' are in the train set as well as in the test set, we will overestimate the algorithm's performance on new documents.

- Diagnosis: patient records coming from the same hospital
  - Hospitals might have different specializations; When we want to predict diagnosis based on the doctors' reports, cancer cases from a clinic specialized in cancer treatments might have higher similarity to each other than cancer cases coming from a orthopaedic hospital. When the reports of the specialized clinic end up in the train set as well as in the test set, we will overestimate the algorithm's capability to correctly classifiy the diagnosis 'cancer'.

6.1.1 recent article summarizing errors when predicting COVID:

Excerpts of the article Hundreds of AI tools have been built to catch covid. None of them helped.: "They looked at 415 published tools and, like Wynants and her colleagues, concluded that none were fit for clinical use." "Both teams found that researchers repeated the same basic errors in the way they trained or tested their tools."

"Many of the problems that were uncovered are linked to the poor quality of the data that researchers used to develop their tools."

- **duplicates:** "Driggs highlights the problem of what he calls Frankenstein data sets, which are spliced together from multiple sources and can contain duplicates."
- **confounding variables:**
  - "Many unwittingly used a data set that contained chest scans of children who did not have covid as their examples of what non-covid cases looked like. But as a result, the AIs learned to identify kids, not covid."
  - "Because patients scanned while lying down were more likely to be seriously ill, the AI learned wrongly to predict serious covid risk from a person's position."
- **different sources:** "In yet other cases, some AIs were found to be picking up on the text font that certain hospitals used to label the scans. As a result, fonts from hospitals with more serious caseloads became predictors of covid risk."
- **human labeling error:** "It would be much better to label a medical scan with the result of a PCR test rather than one doctor's opinion, says Driggs."

6.1.2 How to fix it?

- "Better data would help, but in times of crisis that's a big ask."
- "'Until we buy into the idea that we need to sort out the unsexy problems before the sexy ones, we're doomed to repeat the same mistakes,' says Mateen."

Original articles: Wynants et al., 2020. Prediction models for diagnosis and prognosis of covid-19: systematic review and critical appraisal Roberts et al., 2021. Common pitfalls and recommendations for using machine learning to detect and prognosticate for COVID-19 using chest radiographs and CT scans

6.1.3 Data leakage Literature

- examples of data leakage in competitions: start on page 19
- alternative to the text above here is the video
- Medical data mining: insights from winning two competitions

important: if you're not allowed to read any more 'towardsdatascience' article or 'medium' articles – just remove the cookies for the page (inspect -> applications -> cookies) and reload the page afterwards.

- data leakage when tuning hyper-parameters

1. Morning
    - Data Modelling & Cross Validation
    - data leakage & dependent data
    - **imbalanced data (example in python)**
    - study: Ebanking Fraud
    - Q&A
2. Afternoon
    - Data Basics & historical perspective
    - Linear Regression
    - Trees
    - house prices (regression example in python)
    - Clustering
    - bonus: Hyperparameter Optimization and AutoML
    - Q&A

CHAPTER

SEVEN

CREDITCARD-FRAUD: IMBALANCED DATA

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error, f1_
↪score, accuracy_score,\
                            roc_auc_score, average_precision_score, precision_recall_
↪curve, auc,\
                            roc_curve, precision_score, recall_score

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from copy import copy
from numpy import random

import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'png'
```

```
# https://www.kaggle.com/mlg-ulb/creditcardfraud
df = pd.read_csv("../data/creditcard.csv")
df.shape
```

```
(284807, 31)
```

7.1 the data-set is too large - we subsample the majority class

```
no_fraud = df.index[df['Class']==0]
fraud = df.index[df['Class']==1].tolist()
sampled = random.choice(no_fraud, size=100000, replace=False).tolist()
df = df.loc[fraud + sampled, ].reset_index()
df.to_csv("../data/creditcard_subsampled.csv", header=True, index=False)
```

```
df = pd.read_csv("../data/creditcard_subsampled.csv")
```

```
print(f"num anomalies {np.sum(df['Class']==1)}/{len(df)} = {100*np.sum(df['Class
↪']==1)/len(df):.2f}%")
```

```
num anomalies 492/100492 = 0.49%
```

CHAPTER

EIGHT

WHAT WILL OUR ACCURACY BE AD HOC?

```
pd.crosstab(df.Class.astype(str) + "_true", pd.Series(np.zeros_like(df.Class)).
↪astype(str) + "_Actual")
```

```
col_0   0_Actual
Class
0_true    100000
1_true       492
```

- accuracy $= \frac{\text{true positives + true negatives}}{\text{true negatives + false negatives + true positives + false positives}} = \frac{0+284315}{492+284315} = 0.998$

CHAPTER

NINE

GET A MODEL

```
import lightgbm
model = lightgbm.LGBMClassifier(boosting_type='gbdt', num_leaves=31, max_depth=- 1,␣
↪learning_rate=0.1,
                                n_estimators=500, subsample_for_bin=20000, objective=
↪'binary',
                                subsample=1.0, subsample_freq=0, colsample_bytree=1.0,
                                n_jobs=- 1, silent=True, importance_type='split',
                                is_unbalance = False, scale_pos_weight = 1.0)
```

CHAPTER

TEN

TRAIN / TEST SPLIT

```
X_train = pd.read_csv("../data/creditcard_X_train.csv")
X_test = pd.read_csv("../data/creditcard_X_test.csv")
y_train = pd.read_csv("../data/creditcard_y_train.csv")
y_test = pd.read_csv("../data/creditcard_y_test.csv")
# X, y = df.drop('Class', axis=1), df['Class']
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
clf = copy(model)
clf.fit(X_train,y_train)
# X_train.to_csv("../data/creditcard_X_train.csv", header=True, index=False)
# X_test.to_csv("../data/creditcard_X_test.csv", header=True, index=False)
# y_train.to_csv("../data/creditcard_y_train.csv", header=True, index=False)
# y_test.to_csv("../data/creditcard_y_test.csv", header=True, index=False)
```

```
/home/martin/miniconda3/envs/imbalanced/lib/python3.7/site-packages/sklearn/utils/
↪validation.py:63: DataConversionWarning: A column-vector y was passed when a 1d␣
↪array was expected. Please change the shape of y to (n_samples, ), for example␣
↪using ravel().
  return f(*args, **kwargs)
```

```
LGBMClassifier(is_unbalance=False, n_estimators=500, objective='binary',
               scale_pos_weight=1.0, subsample_for_bin=20000)
```

```
display(y_train.sum(axis=0)[0], y_test.sum(axis=0)[0])
```

```
406
```

```
86
```

```
(clf.predict_proba(X_test.iloc[3:4,:]) + clf.predict_proba(X_test.iloc[3:4,:]))/2
```

```
array([[9.99999999e-01, 1.27675340e-09]])
```

```
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
FPR, TPR, _ = roc_curve(y_test, y_pred_proba)
```

(continues on next page)

(continued from previous page)

```
df_conf = pd.DataFrame(confusion, columns=['non-Fraud','Fraud'], index=['non-Fraud',
 ↪'Fraud'])
df_conf
```

```
           non-Fraud  Fraud
non-Fraud      20012      1
Fraud             23     63
```

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{10.1}$$

$$\text{Recall} = \text{Sensitivity} = \text{TPR} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{10.2}$$

$$\text{F}_1\text{-Score} = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10.3}$$

$$\text{FPR} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{10.4}$$

The normal ROC auc is computed for the True Positive Rate (TPR) and the False Positive Rate (FPR)

```
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
 ↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
 ↪4f}")
print(f"ROC AUC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
Precision 0.984, Recall 0.733
F1 0.84, Accuracy 0.9988
ROC AUC 0.96, AUC PR 0.82
```

```
plt.plot(FPR, TPR, 'b.-', markersize=1, alpha=0.5)
plt.xlabel("FPR")
plt.ylabel("TPR / Recall / Sensitivity")
plt.title(f"credit card $\\bf ROC$ curve")
plt.savefig("/tmp/ROC-curve.png", dpi=200)
plt.show()
```

Because of the imbalance, the FPR has from the beginning on very low values. For imbalanced data, th ROC AUC of this curve is not ideal to compare different algorithms: Suppose there are 100 positive cases in the data and 100'000 negative cases:

$$\text{algorithm 1: 2000 false positives: FPR } = \frac{\text{FP}}{\text{FP + TN}} = \frac{2000}{2000+100000} = 0.0196 \tag{10.5}$$

$$\text{algorithm 2: 200 false positives: FPR } = \frac{\text{FP}}{\text{FP + TN}} = \frac{200}{200+100000} = 0.001966 \tag{10.6}$$

```
plt.hist(y_pred_proba)
```

```
(array([2.0032e+04, 1.0000e+00, 1.0000e+00, 1.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 6.3000e+01]),
 array([1.84737102e-14, 1.00000000e-01, 2.00000000e-01, 3.00000000e-01,
        4.00000000e-01, 5.00000000e-01, 6.00000000e-01, 7.00000000e-01,
        8.00000000e-01, 9.00000000e-01, 1.00000000e+00]),
 <BarContainer object of 10 artists>)
```

```
plt.plot(recall, precision, 'b.-', markersize=1, alpha=0.5)
plt.xlabel("recall")
plt.ylabel("precision")
plt.title(f"credit card $\\bf PR$ curve")
plt.savefig("/tmp/PR-curve.png", dpi=200)
plt.show()
```

For imbalanced data, precision is the better measure: Suppose there are 100 positive cases in the data and 100'000 negative

cases: Both algorithms classifiy 50 cases correctly:

$$\text{algorithm 1: 2000 false positives:} \quad \text{Precision} = \frac{\text{TP}}{\text{TP + FP}} = \frac{50}{50+2000} = 0.0243 \tag{10.7}$$

$$\text{algorithm 2: 200 false positives:} \quad \text{Precision} = \frac{\text{TP}}{\text{TP + FP}} = \frac{50}{50+200} = 0.2 \tag{10.8}$$

```
# keep the workspace tidy
y_pred, confusion, y_pred_proba, precision, recall = None, None, None, None, None
del y_pred, confusion, y_pred_proba, precision, recall
```

CHAPTER

ELEVEN

PROPER TEST SET?

we can also do this with a sklearn data-set splitter:

```
from sklearn.model_selection import StratifiedShuffleSplit
splitter = StratifiedShuffleSplit(n_splits=2, test_size=0.2, train_size=0.8)
idx = next(splitter.split(df, df.Class))
train_idx = idx[0]
test_idx = idx[1]
```

```
len(test_idx)
```

```
20099
```

```
df_train = df.loc[train_idx]
df_test = df.loc[test_idx]
print(f"TRAIN num fraud {np.sum(df_train['Class']==1)}/{len(df_train)} = {100*np.
↪sum(df_train['Class']==1)/len(df_train):.2f}%")
print(f"TEST num fraud {np.sum(df_test['Class']==1)}/{len(df_test)} = {100*np.sum(df_
↪test['Class']==1)/len(df_test):.2f}%")
```

```
TRAIN num fraud 394/80393 = 0.49%
TEST num fraud 98/20099 = 0.49%
```

```
X_train, y_train = df_train.drop('Class', axis=1), df_train['Class']
X_test, y_test = df_test.drop('Class', axis=1), df_test['Class']
clf = copy(model)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
       F   T
F  19993   8
T     22  76
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
```

(continues on next page)

(continued from previous page)

```
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC AUC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
Precision 0.905, Recall 0.776
F1 0.84, Accuracy 0.9985
ROC AUC 0.89, AUC PR 0.83
```

```
# keep workspace tidy
df, idx, df_train, df_test, clf, precision, recall, y_pred, y_pred_proba = None, None,
↪ None, None, None, None, None, None,None
del df, idx, df_train, df_test, clf, precision, recall, y_pred, y_pred_proba
```

CHAPTER

TWELVE

STRATEGIES FOR THE IMBALANCED CASE:

- oversample the minority class
- undersample the majority class
- do both of the former two suggestions
- oversample only the cases that get missclassified
- set `is_unbalance` and/or `scale_pos_weight` parameters in lightgbm

12.1 oversample fraud

12.1.1 Synthetic Minority Oversampling Technique = SMOTE

The original paper is here The idea is to connect very close points in the feature space and generate new samples lying on the connecting line. This is done for the minority class only.

```
import imblearn
from imblearn.over_sampling import SMOTE, ADASYN
X_train_ov, y_train_ov = SMOTE(sampling_strategy=0.1).fit_resample(X_train, y_train)
```

```
clf = copy(model)
clf.fit(X_train_ov,y_train_ov)
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
       F   T
F  19999   2
T      0  98
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC AUC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
Precision 0.980, Recall 1.000
F1 0.99, Accuracy 0.9999
ROC AUC 1.00, AUC PR 1.00
```

```
X_train_ov, y_train_ov, y_pred, y_pred_proba, precision, recall, clf = None, None,␣
↪None, None, None, None, None
del X_train_ov, y_train_ov, y_pred, y_pred_proba, precision, recall, clf
```

12.1.2 Adaptive Synthetic Sampling = ADASYN

The paper can be found here. More synthetic observations are generated in areas where the density of the minority class is very low. Those cases are hard to learn and upsampling them with SMOTE should make it easier to learn.

```
# needs too much RAM:
X_train_ov, y_train_ov = ADASYN(sampling_strategy=0.05).fit_resample(X_train, y_train)
```

```
clf = copy(model)
clf.fit(X_train_ov,y_train_ov)
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
       F   T
F  20001   0
T      2  96
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC AUC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
Precision 0.919, Recall 0.806
F1 0.86, Accuracy 0.9987
ROC AUC 0.97, AUC PR 0.84
```

```
X_train_ov, y_train_ov, y_pred, y_pred_proba, precision, recall, clf = None, None,␣
↪None, None, None, None, None
del X_train_ov, y_train_ov, y_pred, y_pred_proba, precision, recall, clf
```

12.1.3 Borderline-SMOTE

The paper can be found here The principal idea is to upsample only those observations that are not classified correctly. The missclassified instances are found by k-nearest-neighbors.

```
# needs to much RAM for me:
from imblearn.over_sampling import BorderlineSMOTE
X_train_ov, y_train_ov = BorderlineSMOTE(sampling_strategy=0.1).fit_resample(X_train,␣
↪y_train)
```

```
clf = copy(model)
clf.fit(X_train_ov,y_train_ov)
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
       F   T
F  19995   6
T     16  82
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC AUC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
Precision 0.932, Recall 0.837
F1 0.88, Accuracy 0.9989
ROC AUC 0.97, AUC PR 0.84
```

```
X_train_ov, y_train_ov, y_pred, y_pred_proba, precision, recall, clf = None, None,␣
↪None, None, None, None, None
del X_train_ov, y_train_ov, y_pred, y_pred_proba, precision, recall, clf
```

12.1.4 Support-Vector-Machine-SMOTE = SVMSMOTE

The class-boundaries for the Borderline are learned as the support-vectors of a Support-Vector-Machine.

```
from imblearn.over_sampling import SVMSMOTE
X_train_ov, y_train_ov = SVMSMOTE(sampling_strategy=0.1).fit_resample(X_train, y_
↪train)
```

```
clf = copy(model)
clf.fit(X_train_ov,y_train_ov)
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
       F   T
F  19994   7
T     16  82
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
Precision 0.921, Recall 0.837
F1 0.88, Accuracy 0.9989
ROC 0.97, AUC PR 0.86
```

```
# keep the workspace tidy
X_train_ov, y_train_ovm, y_pred, clf = None, None, None, None
del X_train_ov, y_train_ovm, y_pred, clf
```

12.2 weigh minority class

```
clf = copy(model)
clf.set_params(is_unbalance = True)
#, scale_pos_weight
```

```
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
clf.set_params(is_unbalance=False, scale_pos_weight=200.0)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
# house-keeping
y_pred_proba, precision, recall, clf, y_pred = None, None, None, None, None
del y_pred_proba, precision, recall, clf, y_pred
```

12.3 Undersample Fraud

```
from imblearn.under_sampling import RandomUnderSampler
X_train_u, y_train_u = RandomUnderSampler(sampling_strategy=0.5).fit_resample(X_train,
↪ y_train)
print(X_train.shape, X_train_u.shape)
```

```
clf = copy(model)
clf.fit(X_train_u,y_train_u)
y_pred = clf.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC {AUC:.2f}, AUC PR {PR:.2f}")
```

```
y_pred_proba, y_pred, precision, recall, clf, X_train_u, y_train_u = None, None, None,
↪ None, None, None, None
y_pred_proba, y_pred, precision, recall, clf, X_train_u, y_train_u
```

12.4 do both: oversample minority class and undersample majority class

12.4.1 SMOTE and Edited Nearest Neighbors = SMOTE-ENN

- SMOTE is used for upsampling
- ENN removes an observation and its (k=3) nearest neighbors when the majority class of the nearest neighbors is different from the observation's class

12.4.2 SMOTETOMEK

- this method finds reciprocal nearest-neighbors that are in different classes; These pairs are called Tomek-Links
- the observation in such a pair, belonging to the majority class is deleted

```
# not enough RAM
from imblearn.combine import SMOTEENN
X_train_uo, y_train_uo = SMOTEENN().fit_resample(X_train, y_train)
# X_train_uo, y_train_uo = SMOTETomek(random_state=0).fit_resample(X_train, y_train)
# print(X_train.shape, X_train_uo.shape)
```

12.5 get best combination: sampling and hyper-parameters

```
from imblearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import BorderlineSMOTE
clf = copy(model)
clf.set_params(num_leaves=64)
borderline = BorderlineSMOTE(random_state=88)
grid = {'class__n_estimators': [200, 500],
        'class__neg_bagging_fraction': [0.1, 0.3],
        'class__max_depth' : [4, 6, 8,],
        'class__learning_rate' :[0.05, 0.1],
        'sampling__sampling_strategy' :[0.1, 0.3]}
pipeline = Pipeline([('sampling', borderline), ('class', clf)])
grid_cv = GridSearchCV(pipeline, grid, scoring = 'f1', cv = 5, refit=True)

grid_cv.fit(X_train, y_train)
display(grid_cv.best_score_, grid_cv.best_params_)
```

```
pipeline.set_params(**grid_cv.best_params_)
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)#[:, 1]

confusion = confusion_matrix(y_test, y_pred)
df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
y_pred_proba = pipeline.predict_proba(X_test)[:,1] # 2nd column is p(fraud)
AUC = roc_auc_score(y_test, y_pred_proba)
```

(continues on next page)

(continued from previous page)

```
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred):.3f}, Recall {recall_score(y_test,␣
↪y_pred):.3f}")
print(f"F1 {f1_score(y_test, y_pred):.2f}, Accuracy {accuracy_score(y_test, y_pred):.
↪4f}")
print(f"ROC {AUC:.2f}, AUC PR {PR:.2f}")
```

12.6 One-Class SVM

Train on majority class only and classifiy test-set in in-class examples and out-class examples:

```
from sklearn.svm import OneClassSVM

train_normal = X_train[y_train == 0]
train_outliers = X_train[y_train == 1]
```

```
outlier_prop = len(train_outliers) / len(train_normal)
svm = OneClassSVM(kernel='rbf', nu=outlier_prop, gamma=0.000001)
svm.fit(train_normal)
```

```
y_pred = svm.predict(X_test)
y_pred_corrected = np.zeros_like(y_pred)
y_pred_corrected[y_pred == -1] = 1
```

```
y_test
```

```
confusion = confusion_matrix(y_test, y_pred_corrected)

df_conf = pd.DataFrame(confusion, columns=['F','T'], index=['F','T'])
df_conf
```

```
AUC = roc_auc_score(y_test, y_pred_corrected)
precision, recall, _ = precision_recall_curve(y_test, y_pred_corrected, pos_label=1)
PR = auc(recall, precision)
print(f"Precision {precision_score(y_test, y_pred_corrected):.3f}, Recall {recall_
↪score(y_test, y_pred_corrected):.3f}")
print(f"F1 {f1_score(y_test, y_pred_corrected):.2f}, Accuracy {accuracy_score(y_test,␣
↪y_pred_corrected):.4f}")
print(f"ROC {AUC:.2f}, AUC PR {PR:.2f}")
```

Pros:

- It works really well with a clear margin of separation
- It is effective in high dimensional spaces.
- It is effective in cases where the number of dimensions is greater than the number of samples.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Cons:

- It doesn't perform well when we have large data set because the required training time is higher

- It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
- SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of Python scikit-learn library.

1. Morning
    - Data Modelling & Cross Validation
    - data leakage & dependent data
    - imbalanced data (example in python)
    - study: Ebanking Fraud
    - Q&A
2. Afternoon
    - **Data Basics & historical perspective**
    - Linear Regression
    - Trees
    - house prices (regression example in python)
    - Clustering
    - bonus: Hyperparameter Optimization and AutoML
    - Q&A

CHAPTER

THIRTEEN

TITANIC - MACHINE LEARNING FROM DISASTER

13.1 Predict survival on the Titanic and get familiar with ML basics

Home Page

```
   Pclass     Sex   Age Cabin
0       3    male  22.0   NaN
1       1  female  38.0   C85
2       3  female  26.0   NaN
3       1  female  35.0  C123
4       3    male  35.0   NaN
```

13.2 continuous variables

- are always approximations, e.g. the age could be 22 years, 2 months, 1 week, 1 day, 10 hours, 2 minuts, 55 seconds, …
- have an ordering, e.g. $22\,\text{years} < 38\,\text{years}$
- you can interpret differences: $38\,\text{years} - 22\,\text{years} = 16\,\text{years}$

```
    Age
0  22.0
1  38.0
2  26.0
3  35.0
4  35.0
```

13.3 categorical or discrete variables

- are always finite, e.g. Sex is most of the time binary and it's either female or male (I know that's not the best exambple - please do not confuse sex with gender)
- there is no ordering
- you can not interpret differences
- sex is a special case because it's also a **binary variable**; another example for discrete variables is marital status that can be **single, married, widowed, divorced, separated**

```
      Sex
0    male
1  female
2  female
3  female
4    male
```

13.4 ordinal variables

- are always finite, e.g the passenger-class is either 1, 2, or 3 but nothing in between
- it's still possible to have an ordering: p-class 1 $>$ p-class 2 $>$ p-class 3
- you can not interpret the differences: p-class 3 $-$ p-class 2 $=$ p-class 1?

```
   Pclass
0       3
1       1
2       3
3       1
4       3
```

CHAPTER

FOURTEEN

PROCESSING OF VARIABLE

14.1 Continuous Variables

We can easily transform a continuous variable into a ordinal variable by setting cut-points. E.g., we could say that all passengers younger than 2 years are renamed as 'Baby', all passengers between 2 years and 17 years as 'Child', etc..

```
example['Age_binned'] =pd.cut(example.Age,bins=[0,2,17,65,99],labels=['Baby','Child',
↪'Adult','Elderly'])# .iloc[30:40]
example[['Age','Age_binned']][30:40]
```

```
     Age Age_binned
30  40.0      Adult
31   NaN        NaN
32   NaN        NaN
33  66.0    Elderly
34  28.0      Adult
35  42.0      Adult
36   NaN        NaN
37  21.0      Adult
38  18.0      Adult
39  14.0      Child
```

14.2 Categorical or discrete variables

For mathematical models it's hard to work with categories as for example *male*, *female* or *Adult*, *Baby*, *Child*, etc.. This is why we have to turn them into categorical variables. This is done by adding new columns to the data, one for each category-level:

```
nexample = pd.concat([example, pd.get_dummies(example['Age_binned'])], axis=1)
nexample[['Age','Age_binned','Baby', 'Child', 'Adult', 'Elderly']][30:40]
```

```
     Age Age_binned  Baby  Child  Adult  Elderly
30  40.0      Adult     0      0      1        0
31   NaN        NaN     0      0      0        0
32   NaN        NaN     0      0      0        0
33  66.0    Elderly     0      0      0        1
34  28.0      Adult     0      0      1        0
35  42.0      Adult     0      0      1        0
36   NaN        NaN     0      0      0        0
37  21.0      Adult     0      0      1        0
```

(continues on next page)

(continued from previous page)

```
38  18.0      Adult     0      0      1        0
39  14.0      Child     0      1      0        0
```

This is called:

- one-hot encoding
- sometimes this is also incorrectly called dummy encoding
- real **dummy encoding** has one column less than one-hot encoding: The idea is, if its not *Child*, nor *Adult* or *Elderly*, then is must be *Baby* - so we do not need an extra column for *Baby*

Most intuitively the **real** dummy-encoding can be seen with **sex**: even though there are two different categories, we just need one column

```
      Sex  Sex2
0    male     0
1  female     1
2  female     1
3  female     1
4    male     0
```

14.3 ordinal data

- there are methods for ordinal data, e.g. ordinal regression
- most of the time ordinal variables are just treated as categorical variables

CHAPTER

FIFTEEN

MISSING DATA

Data can be missing:

- at random
- systematically, i.e. the fact that the data is missing could bear some valuable information

For categorical and ordinal data, missing data is just another category. For continuous variables there are several possibilities to deal with missing data. The most frequent ones are:

- imputation by the mean (the mean-value of all non-missing values is taken)
- imputation by the median (the value with half of all values larger and half of all values smaller is taken)
- imputation by the mode (the most frequent value is taken)

However, since we do not know for sure, why data is missing it is often helpfull to keep track of it by creating a new indicator variable for missing values:

First, we create the indicator variable:

```
     Age  missing_Age
30  40.0            0
31   NaN            1
32   NaN            1
33  66.0            0
34  28.0            0
35  42.0            0
36   NaN            1
37  21.0            0
38  18.0            0
39  14.0            0
```

Second, we impute missing values with the average Age:

```
          Age  missing_Age
30  40.000000            0
31  29.699118            1
32  29.699118            1
33  66.000000            0
34  28.000000            0
35  42.000000            0
36  29.699118            1
37  21.000000            0
38  18.000000            0
39  14.000000            0
```

15.1 Interactions

Interactions are another important concept in linear modelling. Here, the effect of one variable on the dependent variable y depends on the value of another variable. In the example below we try to model the probability that a person buys a house. Of course, monthly income is an important variable and the higher it is, the more likely that said person will buy a house. Another important variable is marital status. Married people with children in the household tend strongly to buy houses, especially if their monthly income is high. On the other hand, singles, even if they have a high income, will tend not to buy a house. So we see, the variable "monthly income" **interacts** with the variable "marital status": the effect of the two variables together is more than the sum of the effects of the individual variables.

15.2 Standardization / Normalization

Sometimes we have to bring different variables into the same range. This is very important for Neural Networks, but also for other algorithms it can sometimes be benefitial. Assume, we have the age of some passengers as in the following table:

We obtain normalized values by applying the following z-transform:

$$z_i = \frac{x_i - \bar{x}}{\sigma} \tag{15.1}$$

$$\text{with:} \tag{15.2}$$

$$\bar{x} = \text{mean} \tag{15.3}$$

$$\sigma = \text{standarddeviation} \tag{15.4}$$

```
passenger_age['normalized age'] = (passenger_age['Age'] - passenger_age['Age'].
 ↪mean())/passenger_age['Age'].std()
display(passenger_age)
```

```
                                                Name   Age  normalized age
0                              Braund, Mr. Owen Harris  22.0       -0.427976
```

(continues on next page)

(continued from previous page)

```
1   Cumings, Mrs. John Bradley (Florence Briggs Th...  38.0        0.481000
2                              Heikkinen, Miss. Laina  26.0       -0.200732
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0        0.310567
4                            Allen, Mr. William Henry  35.0        0.310567
6                             McCarthy, Mr. Timothy J  54.0        1.389976
7                      Palsson, Master. Gosta Leonard   2.0       -1.564197
8   Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)  27.0       -0.143921
9                 Nasser, Mrs. Nicholas (Adele Achem)  14.0       -0.882465
10                    Sandstrom, Miss. Marguerite Rut   4.0       -1.450575
11                            Bonnell, Miss. Elizabeth  58.0        1.617220
12                     Saundercock, Mr. William Henry  20.0       -0.541598
13                        Andersson, Mr. Anders Johan  39.0        0.537811
14               Vestrom, Miss. Hulda Amanda Adolfina  14.0       -0.882465
15                   Hewlett, Mrs. (Mary D Kingcome)   55.0        1.446787
```

Now, let's do the same for another variable: Fare - the price payed for the passage

```
                                                Name   Age  normalized age  \
0                             Braund, Mr. Owen Harris  22.0       -0.427976
1   Cumings, Mrs. John Bradley (Florence Briggs Th...  38.0        0.481000
2                              Heikkinen, Miss. Laina  26.0       -0.200732
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0        0.310567
4                            Allen, Mr. William Henry  35.0        0.310567
6                             McCarthy, Mr. Timothy J  54.0        1.389976
7                      Palsson, Master. Gosta Leonard   2.0       -1.564197
8   Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)  27.0       -0.143921
9                 Nasser, Mrs. Nicholas (Adele Achem)  14.0       -0.882465
10                    Sandstrom, Miss. Marguerite Rut   4.0       -1.450575
11                            Bonnell, Miss. Elizabeth  58.0        1.617220
12                     Saundercock, Mr. William Henry  20.0       -0.541598
13                        Andersson, Mr. Anders Johan  39.0        0.537811
14               Vestrom, Miss. Hulda Amanda Adolfina  14.0       -0.882465
15                   Hewlett, Mrs. (Mary D Kingcome)   55.0        1.446787

       Fare  normalized_Fare
0    72.500        -0.868610
1   712.833         2.347294
2    79.250        -0.834709
3   531.000         1.434086
4    80.500        -0.828432
6   518.625         1.371936
7   210.750        -0.174285
8   111.333        -0.673581
9   300.708         0.277505
10  167.000        -0.394008
11  265.500         0.100682
12   80.500        -0.828432
13  312.750         0.337983
14   78.542        -0.838265
15  160.000        -0.429164
```

The un-standardized and standardized variables accross passengers look like this:

```
<AxesSubplot:>
```

Age
Fare

normalized age
normalized_Fare

Before Normalization
Age
Fare
Fare

After Normalization
Age
Fare
Fare

CHAPTER

SIXTEEN

NOT COVERED HERE

The following topics are more advanced and do not apply to tree-methods. In a possible follow-up we can discuss them as well:

- power-transforms
- mean-encoding

1. Morning
    - Data Modelling & Cross Validation
    - data leakage & dependent data
    - imbalanced data (example in python)
    - study: Ebanking Fraud
    - Q&A
2. Afternoon
    - Data Basics & historical perspective
    - **Linear Regression**
    - Trees
    - house prices (regression example in python)
    - Clustering
    - bonus: Hyperparameter Optimization and AutoML
    - Q&A

CHAPTER

SEVENTEEN

KNOWLEDGE DISCOVERY -> DATA MINING -> DATA SCIENCE -> MACHINE LEARNING

CHAPTER

EIGHTEEN

LINEARE REGRESSION

In der nachfolgenden Zelle werden zuerst Daten geladen, die zur Veranschaulichung der linearen Regression dienen. Anschliessend wird ein lineares Modell mit Hilfe der der Klasse Lineare Regression aus `sklearn.linear_model` gerechnet. Die Vorhersage (d.h. die Geradengleichung) ergibt sich aus den Koeffizienten durch $y = a + bX$.

Der Plot zeigt die berechnete Regressionsgerade, sowie die Abweichungen (die Fehler) der wirklichen Messwerte von dieser Geraden. Diese Abweichungen werden als **Residuen** bezeichnet, weil es der Anteil der gemessenen Werte ist, der "übrig bleibt", d.h. nicht durch das Modell erklärt werden kann. Vorhergesagte Variablen werden meist mit einem Dach (Hut) bezeichnet, sowie $\hat{y}$.

CHAPTER

NINETEEN

MULTIVARIATE CASE: MORE THAN ONE X VARIABLE

Für Multivariate Lineare Regression kann die Schreibweise mit Matrizen zusammengefasst werden. Dafür kann es lohnend sein, sich die Matrizen-Multiplikation noch einmal kurz anzusehen.

$$
\begin{aligned}
y_1 &= a + b_1 \cdot x_{11} + b_2 \cdot x_{21} + \cdots + b_p \cdot x_{p1} \\
y_2 &= a + b_1 \cdot x_{12} + b_2 \cdot x_{22} + \cdots + b_p \cdot x_{p2} \\
&\cdots\cdots \\
y_i &= a + b_1 \cdot x_{1i} + b_2 \cdot x_{2i} + \cdots + b_p \cdot x_{pi}
\end{aligned}
$$

$$
\begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_i \end{bmatrix} = a + \begin{bmatrix} x_{11} & x_{21} & x_{31} & \dots & x_{p1} \\ x_{12} & x_{22} & x_{32} & \dots & x_{p2} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ x_{1i} & x_{2i} & x_{3i} & \dots & x_{pi} \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_p \end{bmatrix}
$$

Den konstanten inercept Term (a) können wir mit in den Vektor der Parameter **b** aufnehmen, indem wir in **X** eine Einser-Spalte hinzufügen. Somit wird die Schreibweise sehr kompakt und der intercept a wird nicht mehr explizit aufgeführt:

$$
\begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_i \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{21} & x_{31} & \dots & x_{p1} \\ 1 & x_{12} & x_{22} & x_{32} & \dots & x_{p2} \\ & \dots & \dots & \dots & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 1 & x_{1i} & x_{2i} & x_{3i} & \dots & x_{pi} \end{bmatrix} \cdot \begin{bmatrix} a \\ b_1 \\ b_2 \\ . \\ . \\ b_p \end{bmatrix}
$$

In Matrizen-Schreibweise können wir jetzt einfach schreiben: $\mathbf{y} = \mathbf{Xb}$

CHAPTER

TWENTY

POLYNOMIAL REGRESSION AS AN EXAMPLE FOR MORE THAN ONE VARIABLE

Um einfach Multivariate Lineare Regression an einem Beispiel zeigen zu können wird die quadratische Regression (ein Spezial-Fall der Multivariaten Regression) eingeführt. Eine neue Variable entsteht durch das Quadrieren der bisherigen univiaraten Variable x. Das Praktische ist, dass sich der Sachverhalt der Multivariaten Regression noch immer sehr schön 2-dimensional darstellen lässt. $y = a + b_1 x + b_2 x^2$

Hier ist zu beachten:

- wir haben jetzt zwei Variablen und können folglich unsere Formel in Matrizen-Schreibweise anwenden
- mehr Variablen führen hoffentlich zu einem besseren Modell
- durch den quadratischen Term ist die resultierende Regressions-Funktion keine Gerade mehr. **Der Ausdruck "linear" in Linearer Regression bedeutet dass die Funktion linear in den Parametern $a, \mathbf{b_1}, \mathbf{b_2}$ ist. Für alle Werte einer Variablen $\mathbf{x_1}$ gilt der gleiche Parameter $\mathbf{b_1}$. Es bedeutet nicht, dass die Regressions-Funktion durch eine gerade Linie gegeben ist!**
- ausserdem bedienen wir uns hier eines Tricks: Die Variable x^2 müsste eigentlich eine eigene Achse bekommen. Dann wäre die Regressions-Gerade wieder eine gerade Linie - nur lässt sich das leider nicht mehr schön darstellen.

Nachfolgend fügen wir die weitere Variable durch Quadrieren der bisherigen Variable hinzu und berechnen abermals das Lineare Modell aus `sklearn.linear_model`.

```
(-5.0, 110.77315979942053)
```

20.1 Overfitting

Nun wird diese Vorgehensweise für weitere Terme höherer Ordnung angewendet. Graphisch lässt sich zeigen, dass die Anpassung des Modells an die Daten immer besser wird, die Vorhersage für **neue Datenpunkte** aber sehr schlecht sein dürfte. Man sagt dann, das Model **"generalisiert"** sehr schlecht. Das Polynom hat an vielen Stellen Schlenker und absurde Kurven eingebaut. Dies ist ein erstes Beispiel für **"overfitting"**.Einen 'perfekten' fit erhält man, wenn man genau so viele Paramter (10 Steigunskoeffizienten + intercept) hat wie Daten-Messpunkte.

The important points to note here:

- the fit to our empirical y-values gets better
- at the same time, the regression line starts behaving strangly
- the predictions made by the regression line in between the empirical y-values are grossly wrong: this is an example of **overfitting**

```
(-10.0, 115.77315979942053)
```

20.2 perfect fit: as many variables as data samples

A perfect fit is possible as is demonstrated next. We have as many variables (terms derived from x) as observations (data points). So for each data point we have a variable to accommodate it. **Note**, that a perfect fit is achieved with 10 variables + intercept. The intercept is also a parameter and in this case the number of observations n equals the number of variables p, i.e. $p = n$.

```
(-10.0, 125.77315979942053)
```

20.3 Bias-Variance Tradeoff

Wiki

- `Bias`: Underfitting
- `Variance`: Overfitting - the model overfits pecularities of the data sample

This is the perspectiv of classical statistics:

- more parameters lead to overfitting
- the results of models with many parameters are not reliable (due to the high variance)
- with more parameters it's harder for single parameters to reach the significance threshold (statistical testing)
- smaller models are better (epistemology: prefer simpler solutions if they are as good as the more complex ones)

But neural networks are heavily over-parameterized with far more weight-parameters than independent samples in the training data. How comes they generalize quite well?

Following Belkin et al., 2019 and Dar et al., 2021:

- When we have as many parameters as data samples, the number of solutions is very constrained. The model has to "stretch" to reach the interpolation threshold with a limited capacity. This explains the weird loops the polynomial makes.
- When we have more parameters than data points the space of interpolating solutions opens-up, actually allowing optimization to reach lower-norm interpolating solutions. These tend to generalize better, and that's why you get the second descent on test data.

For the interested reader:

- On the Bias-Variance Tradeoff: Textbooks Need an Update
- Double Descent
- Deep Double Descent
- Are Deep Neural Networks Dramatically Overfitted?

CHAPTER

TWENTYONE

DEALING WITH OVERFITTING

Wie wir gesehen haben tendiert klassische Lineare Regression zu 'overfitting' sobald es wenige Datenpunkte gibt und mehrere Koeffizienten berechnet werden. Eine Lösung für dieses Problem ist, die Koeffizienten $b_1, b_2, b_3, ...$ kleiner zu machen. Dies kann erreicht werden, wenn der Fehler der Regression mit grösseren Koeffizienten auch grösser wird. Um nun das Minimum der Fehlerfunktion zu finden ist ein probates Mittel, die Koeffizienten kleiner zu machen und somit implizit 'overfitting' zu verhindern. Parameter können jetzt nur noch sehr gross werden, wenn dadurch gleichzeitig der Fehler stark reduziert werden kann.

Nachfolgend wird ein Strafterm ('penalty') für grosse Parameter eingeführt. Im Falle der Ridge-Regression gehen die Koeffizienten quadriert in die Fehlerfunktion mit ein. Der Gewichtungsfaktor λ bestimmt die Höhe des Strafterms und ist ein zusätzlicher Parameter für den – je nach Datensatz – ein optimaler Wert gefunden werden muss.

21.1 Ridge regression

Remember this formula:

$$\sum_i^n (y_i - \hat{y_i})^2 = \sum_i^n [y_i - (a + b \cdot x_i)]^2 \tag{21.1}$$

To make the error term larger for extrem values of b, we could simply add $\lambda \cdot b^2$ to the error:

$$\sum_i^n (y_i - \hat{y_i})^2 + \lambda b^2 = \sum_i^n [y_i - (a + b \cdot x_i)]^2 + \lambda b^2 \tag{21.2}$$

The parameter λ is for scaling the amount of shrinkage. Die beiden Ausdrücke

$$\sum_i^n [y_i - (a + b \cdot x_i)]^2 \tag{21.3}$$

$$\lambda b^2 \tag{21.4}$$

sind wie Antagonisten. Der Koeffizient b darf nur gross werden, wenn er es vermag (21.3) stark zu verkleinern, so dass der Zugewinn in (21.3) den Strafterm in (21.4) überwiegt.

For two variables we can write:

$$\sum_i^n (y_i - \hat{y_i})^2 + \lambda b_1^2 + \lambda b_2^2 = \sum_i^n [y_i - (a + b_1 \cdot x_{i1} + b_2 \cdot x_{i2})]^2 + \lambda b_1^2 + \lambda b_2^2$$

And in matrix notation for an arbitrary number of variables:

$$\min = (\mathbf{y} - \hat{\mathbf{y}})^2 + \lambda \mathbf{b}^2 = (\mathbf{y} - \mathbf{Xb})'(\mathbf{y} - \mathbf{Xb}) + \lambda \mathbf{b}'\mathbf{b}$$

21.1.1 example of ridge regression

Next, we will apply ridge regression as implemented in the python `sklearn` library and compare the results to the linear algebra solution. Note, that we have to center the variables.

- we can center **X** and **y** and display the result in the centered coordinate system
- or we can center **X** and add the mean of **y** to the predicted values to display the result in the original coordinate system. This approaches allows for an easy comparison to the overfitted result

Die Zeile `Xc = X - np.mean(X, axis=0)` standardisiert die Variablen auf den Mittelwert von 0

Now, it becomes clear why Ridge Regression was invented before Lasso Regression. We have a analytical solution. Ridge is nearer to 'old school statistics' than Lasso is.

21.2 Lasso

Alternativ zu einem quadratischen Strafterm b^2 könnte man auch den absoluten Wert nehmen $|b|$. In diesem Fall erhält man die sog.~Lasso Regression; $\lambda \cdot |b|$ wird zum Vorhersage-Fehler addiert:

$$\sum_i^n (y_i - \hat{y}_i)^2 + \lambda|b| = \sum_i^n [y_i - (a + b \cdot x_i)]^2 + \lambda|b|$$

Für zwei Variablen würde man folglich schreiben:

$$\sum_i^n (y_i - \hat{y}_i)^2 + \lambda|b_1| + \lambda|b_2| = \sum_i^n [y_i - (a + b_1 \cdot x_{i1} + b_2 \cdot x_{i2})]^2 + \lambda|b_1| + \lambda|b_2|$$

Leider gibt es im Gegesatz zur Ridge Regression keine eindeutige analytische Lösung um die Koeffizienten der Lasso Regression zu erhalten. Hier kommen iterative Verfahren zum Einsatz, wie wir sie in Session 2 kennen lernen werden. Iterative Verfahren haben sich erst sehr spät durchgesetzt - nicht zuletzt wegen der Rechenleistung die sie benötigen.

21.2.1 kurzer Einschub: klassische Statistik vs. Machine Learning

- Mathematisch liess sich lange Zeit nur ein lineares Gleichungssystem zuverlässig lösen (Rechenpower). Deshalb wurde Ridge-Regression auch vor Lasso-Regression erfunden. Für ersteres Verfahren gibt es eine analytische Lösung.
- Das lineare Modell setzt voraus, dass alle Variablen darin voneinander unabhängig und normal verteilt sind. Dies trifft auf fast keinen Umstand in unserer Welt zu.
- Konfidenzintervalle und Signifikanzen sind das direkte Resultat dieser Annahmen und der damit verbundenen mathmatischen Lösung - der Inversion der Kreuzprodukt-Matrix - so wie wir das besprochen haben.
- **"Overfitting"** ist der Begriff, der verwendet wurde, wenn das verwendete mathematische Verfahren die Daten der Stichprobe zu genau repräsentiert und auf neue Daten schlecht generalisiert.
- Leider wurde **"overfitting"** oft gleichbedeutend mit zu vielen Variablen verwendet.
- Wird die Grösse der Parameter (die Norm) klein gehalten (Ridge, Lasso) so tritt **"overfitting"** nicht auf.
- Mittlerweile gibt es zuverlässige Verfahren, die overfitting zu verhindern wissen. Da die Modellannahmen in den Wissenschaften oft nur getroffen wurden, weil es für diese eine analytische Lösung gibt, müssten eigentlich viele Lehrbücher umgeschrieben werden.

21.2.2 kurzer Einschub: klassische Statistik vs. Machine Learning

- Die Verfahren mit vielen Variablen finden in vielen Anwendungen sehr gute Lösungen und haben einige Anwedungsfelder geradezu revolutioniert (Sprach- und Bilderverarbeitung).
- Wissenschaftstheoretisch sind die neuen Verfahren nicht hinreichend, aber auch die alten Verfahren sind von geringem Nutzen, wenn die Annahmen falsch sind.
- Es wird Zeit, bisherige klassische Statistik und Verfahren des maschinellen Lernen miteinander zu versöhnen. Eine neuere, umfassende Theorie muss entwickelt werden.

lasso regression for polynome of 7th degree and $\lambda = 2$
lasso regression
unregularized regression

CHAPTER

TWENTYTWO

EXTENSION: LOGISTIC REGRESSION AND THE GLM

Es gibt andere Modelle, die eng verwandt mit der hier besprochenen Linearen Regression sind. Das Prominenteste unter ihnen ist die **Logistische Regression**. Diese Modell gehört zu dem "**Verallgemeinerten Linearen Modell**" (im engl. **generalized lineare model** (GLM)). Diese Modelle dürfen nicht mit dem "**Allgemeinen Linearen Modell**" (im engl. **general linear model**) verwechselt werden. Letzteres parametrisiert eine Varianzanalyse als ein lineares Modell mit Dummy-Variablen. Das Verallgemeinerte Lineare Modell erweitert die Lineare Regression um Modelle, deren Fehler nicht normalverteilt sind. Dieser Artikel in der Wikipedia gibt weitere Auskunft.

22.1 exponential family of distributions

Aus der Perspektive der Modernen Statistik beinhaltet das Verallgemeinerte Lineare Modell verschiedene Lineare Modelle, unter anderem das der klassischen linearen Regression. Eine Verteilung, die in der "exponential family" von Verteilungen ist, kann immer folgendermassen geschrieben werden:

$$f(y|\theta) = \exp\left(\frac{y\theta + b(\theta)}{\Phi} + c(y, \Phi)\right), \tag{22.1}$$

wobei θ als Kanonischer Parameter bezeichnet wird, welcher eine Funktion von μ ist dem Mittel. Diese Funktion wird als Kanonische Link-Funktion bezeichnet. Wie wir später an einem Beispiel sehen werden, ist es genau diese Funktion welche die Beziehung zwischen der abhängigen Variablen und den unabhängigen Variablen linearisiert. Der Vollständigkeit halber: $b(\theta)$ ist eine Funktion des Kanonischen Parameters und ist somit ebenfalls von μ abhängig. Φ wird als Streuungsparameter bezeichnet und $c(y, \Phi)$ ist eine Funktion, die sowohl von beobachteten Daten wie auch dem Streuungsparameter abhängig ist.

22.1.1 Normalverteilung

$$\begin{aligned} f(y|\mu,\sigma) = & \quad (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\frac{y^2-2y\mu+\mu^2}{\sigma^2}\right) \\ = & \quad \exp\left(\frac{y\mu-\frac{\mu^2}{2}}{\sigma^2} - \frac{1}{2}\left(\frac{y^2}{\sigma^2} + \log(2\pi\sigma^2)\right)\right), \quad \text{wobei} \end{aligned}$$

$\mu = \theta(\mu)$, d.h. μ ist der Kanonische Parameter und die Link-Funktion ist die Identitäts-Funktion. Der Mittelwert kann also ohne weitere Transformation direkt modelliert werden, so wie wir es in der klassischen Linearen Regression machen. Der Streuungsparameter Φ ist durch σ^2, die Varianz gegeben. Dies ist die klassische Lineare Regression normalverteilter Variablen

22.1.2 Poisson distribution

Die Poisson-Verteilung gehört ebenfalls der exponential family von Verteilungen an:

$$\begin{aligned} f(y|\mu) = & \quad \frac{\mu^y e^{-\mu}}{y!} = \mu^y e^{-\mu} \frac{1}{y!} \\ = & \quad \exp\left(y \log(\mu) - \mu - \log(y!)\right) \end{aligned}$$

Die Link-Funktion ist hier $\log(\mu)$. Beachte bitte, dass die Poisson-Verteilung keinen Streuungsparameter besitzt.

22.1.3 Bernoulli distribution ⇒ logistic regression

Zuguter Letzte, die Bernoulli Verteilung, von der wir die Logistische Regression ableiten können. Die Bernoulli Verteilung eignet sich um binäre Ereignisse zu modellieren, die sich gegenseitig ausschliessen. Ein klassisches Beispiel ist der wiederholte Münzwurf. Die Wahrscheinlichkeit für 'Kopf' wird mit π bezeichnet, dir für 'Zahl' mit $(1-\pi)$. Hiermit lässt sich die Wahrscheinlichkeit berechnen, mit einer fairen Münze bei 10 Würfen eine bestimmte Sequenz mit genau 7 Mal 'Kopf' zu erhalten:

$$\pi^7(1-\pi)^3 = 0.5^7 0.5^3 = 0.5^{10} = 0.0009765625 \tag{22.2}$$

Vorsicht, wenn wir die Wahrscheinlichkeit für Sequenzen mit genau 7 Mal Kopf berechnen wollen, benötigen wir noch den Binomial-Koeffizienten, der uns die Anzahl an möglichen Sequenzen mit 7 Mal 'Kopf' angibt.

Jetzt zeige ich, wie wir die Bernoulli Verteilung so umschreiben können, dass man ihre Zugehörigkeit zur exponential family von Verteilungen erkennt:

$$\begin{aligned} f(y|\pi) = & \quad \pi^y(1-\pi)^{1-y} = \exp\left(y\log(\pi) + (1-y)\log(1-\pi)\right) \\ = & \quad \exp\left(y\log(\pi) + \log(1-\pi) - y\log(1-\pi)\right) \\ = & \quad \exp\left(y\log(\tfrac{\pi}{1-\pi}) + \log(1-\pi)\right), \quad \text{wobei} \end{aligned}$$

sich die Link-Funktion zu $\log(\frac{\pi}{1-\pi})$ ergibt. Diese Funktion wird auch als Logit-Funktion bezeichnet. Die Umkehrfunktion der Logit-Funktion ist die **Logistische Funktion**. Es ist also die Logit-Funktion, die als lineare Kombination der unabhängigen Variablen modelliert wird. $\log(\frac{\pi}{1-\pi}) = a + b_1x_1 + ... + b_jx_j$. Wenn wir den rechten Teil dieser Gleichung in die Logistische Funktion einsetzen erhalten wir die geschätzten Wahrscheinlichkeiten:

$$P(y=1|x) = \frac{\exp(a + b_1x_1 + ... + b_jx_j)}{1 + \exp(a + b_1x_1 + ... + b_jx_j)}. \tag{22.3}$$

CHAPTER

TWENTYTHREE

NEURAL NETWORK

Es ist auch möglich Neuronale Netzwerke unter dem Blickwinkel der Linearen Regression zu betrachten. Ein Netzwerk mit nur einer Eingabe-Schicht und einem Neuron wird als Perceptron bezeichnet. Die Aktivierungs-Funktion dieses Neurons ist entweder die Identitäts-Funktion, so wie in der klassischen Linearen Regression oder die Logistische Funktion wie in der Logistischen Regression. In letzterem Fall soll das Perceptron Wahrscheinlichkeiten für binäre Ereignisse bestimmen.

23.1 classical linear regression

Im Jargon der neural network community werden unsere b-Koeffizienten als **Gewichte** bezeichnet. Der intercept α heisst **bias**. Erinnert Euch, dass wir den intercept α in den Vektor $\boldsymbol{\beta}$ der b-Koeffizienten aufgenomen haben, indem wir eine Einser-Spalte in die Variablen-Matrix $\mathbf{X}$ eingefügt haben. Wir konnten also Schreiben:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$$

In der obigen Graphik könnt ihr sehen, dass im Perceptron die Input-Variablen mit den Gewichten der Verbindungen multipliziert werden und dass der konstante Wert α hinzu addiert wird. Wie in der Linearen Regression werden diese Produkte dann aufsummiert.

Im Kontext Neuronaler Netzwerke wird der Vektor $\boldsymbol{\beta}$ als Netzwerk-Gewichte bezeichnet und wird mit $\mathbf{W}$ angegeben. Wir hatten gelernt, dass Vektoren mit kleinen Buchstaben bezeichnet werden. In einem richtigen Neuronalen Netz haben wir in einer Schicht viel Perceptrons nebeneinander. Alle erhalten aber den Input aus der darunter liegenden Schicht. Fügt man die Gewichts-Vektoren der einzelnen Neurone in eine Matrix zusammen, erhält man $\mathbf{W}$. Neuronale Netzwerke sind also eigentlich nur viele parallele und hintereinander geschaltete Regressionen, die sehr effizient mit Matrizen-Multiplikation gerechnet werden können.

23.2 logistic regression

Für die logistische Aktivierungs-Funktion schreiben wir:

$$P(y = 1|x) = \frac{\exp(a + b_1 x_1 + ... + b_j x_j)}{1 + \exp(a + b_1 x_1 + ... + b_j x_j)}$$

Diese Funktion nähert sich asymptotisch der 0 für sehr kleinen Werte und der 1 für sehr grosse Werte.

23.2.1 Weight decay

In der Literatur zu Neuronalen Netzwerken wird der l_2 Strafterm als "weight decay" bezeichnet. Dieser Strafterm ist Teil des optimizers und nicht der einzelnen Neurone. Wie auch für Ridge Regression wird weight in die Fehler-Funktion mit aufgenommen:

$$L' = L + \lambda \sum_i w_i^2,$$

mit L als Loss (oder Fehler) und den w_i als die Gewichte der eingehenden Verbindungen der Neurone.

1. Morning
   - Data Modelling & Cross Validation
   - data leakage & dependent data
   - imbalanced data (example in python)
   - study: Ebanking Fraud
   - Q&A
2. Afternoon
   - Data Basics & historical perspective
   - Linear Regression
   - **Trees**
   - house prices (regression example in python)
   - Clustering
   - bonus: Hyperparameter Optimization and AutoML
   - Q&A

CHAPTER

TWENTYFOUR

WHY TREES?

TABLE 10.1. *Some characteristics of different learning methods. Key:* ▲*= good,* ◆*=fair, and* ▼*=poor.*

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

image taken form p.370

In nachfolgender Graphik wird demonstriert, wie ein decision-tree classifier nach und nach den Input-Variablen-Raum unterteilt um möglichst reine Unterräume zu erhalten. Diese Unterräume entsprechen den jeweiligen Knoten im Baum (nodes), bzw. den Blättern. Wichtig ist, dass diese splits auf einer Variablen immer **orthogonal** sind.

```
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
```

Dies ist der decision-trees mit den splits, wie sie oben dargestellt sind.

```
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
```

24.1 Compare to linear Regression (logistic regression in this case)

Logistic Regression only works for binary classes. But we can always classify __o__ne class __v__ersus the __r__est (ovr) of the other classes - this allows multiclass classification with logistic regression. We can see, whereas the classification tree can approximate **non-linear seperating lines** with many rectangular splits, the logistic regression can only do linear splits.

Classifier tree depth 5, training accuracy=96.67%
20
15
10
5
0
−5
x_2
−4
−2
0
2
4
x_1

24.2 Splitting criteria

For most variants of classification trees, there are basically two important splitting statistics:

1. Gini Impurity
2. Entropy

Gini Impurity

$$\text{Gini} = \sum_{i}^{n} p_i(1 - p_i) \tag{24.1}$$

For a binary classification problem (either 0 or 1) the Gini-Index is:

$$\text{Gini} = p_1(1 - p_1) + p_2(1 - p_2) \tag{24.2}$$

Here, p_1 is the purity of the respective node in the tree.

At the root of the tree, we have the following impurities for class 1 and class 0:

$$\begin{aligned} p_{01} = & \quad \tfrac{40}{120} = 0.3333 \\ p_{00} = & \quad \tfrac{80}{120} = 0.6666 \end{aligned}$$

The Gini-Impurity at the root is given by:

$$\text{Gini}_0 = p_{01} \cdot (1 - p_{01}) + p_{00} \cdot (1 - p_{00}) = 0.4444 \tag{24.3}$$

Next we compute the Gini-Impurities Gini_1 and Gini_2 for the child-nodes after the first split:

$$\begin{aligned} p_{11} = & \quad \tfrac{28}{70} = 0.4 \\ p_{10} = & \quad \tfrac{42}{70} = 0.6 \end{aligned}$$

The Gini-Impurity of the first child node is given by:

$$\text{Gini}_1 = p_{11} \cdot (1 - p_{11}) + p_{10} \cdot (1 - p_{10}) = 0.48 \tag{24.4}$$

For the second child node, we get:

$$\begin{aligned} p_{21} = & \quad \tfrac{12}{50} = 0.24 \\ p_{20} = & \quad \tfrac{38}{50} = 0.76 \end{aligned}$$

The Gini-Impurity of the second child node is given by:

$$\text{Gini}_2 = p_{21} \cdot (1 - p_{21}) + p_{20} \cdot (1 - p_{20}) = 0.3648 \tag{24.5}$$

In the left child node, there are 70 observations, whereas in the right child node, we only have 50 observations. To compute the overall Gini-impurity after the first split, we have the weight the Gini-Impurities of the two child-nodes with the fraction of observations they represent:

$$\text{Gini}_{\text{split1}} = \frac{70}{120} \cdot 0.48 + \frac{50}{120} \cdot 0.3658 = 0.4320 \tag{24.6}$$

Instead of Gini-Impurity, we could just take classification error as a criterion - this seems most intuitive: The classification error in the root node is given by:

$$p_0 = \frac{40}{120} = \mathbf{0.333} \tag{24.7}$$

The classification error in the first child-node is:

$$p_1 = \frac{28}{70} = 0.4 \tag{24.8}$$

And the classification error in the second child-node is given by:

$$p_2 = \frac{12}{50} = 0.24 \tag{24.9}$$

Now, to compute the reduction in classifiication-error, we have again to weigh the two nodes by the number of observations the contain:

Classification-error after the first split is:

$$\frac{70}{120} \cdot 0.4 + \frac{50}{120} \cdot 0.24 = \mathbf{0.333} \tag{24.10}$$

CHAPTER

TWENTYFIVE

GENETIC ALGORITHMS

Hidden Order: How Adaptation Builds Complexity

John H Holland, Professor of Psychology and of Electrical Engineering and Computer Science ,

Basic Books, 21.08.1995 - 185 Seiten

★★★★★

0 Rezensionen ⓘ

The father of the field of genetic algorithms, and one of the pioneers of the new science of comp
carries on every page the weight of Holland's authority and distinctive point of view. This book e
description of what we might do to enhance our theoretical understanding of cas. He suggests v

25.1 Evolutionary Decision Trees:

Genetic Algorithms try to mimic the genetic recombination happening in sexual reproduction:

- Crossing Over: random recombination between the paired chromosomes inherited from each of one's parents, generally occurring during meiosis (wiki);
- fertilization: haploid chromosomes from a random mother and a random father form a new diploid set of chromosomes
- mutation: randomly, some genes may change accidentally

Survival of the Fittest:

- only a certain number of the offspring passes the evolutionary bottleneck (the best adapted ones + some randomness)
- the survivors form the next parent generation with probability proportional to their fitness

Applied to decision trees on can:

- start growing a generation of decision trees, and recombine the fittest trees
- grow and prune decision trees with evolutionary principles

25.2 my opinion about evolutionary algorithms:

Before the advent of modern machine-learning algorithms, most algorithms (classification-trees) where optimized in a hill-climbing fashion: **straight to the top** But as we all know, the seemingly shortest path is not necessarily the best one. The straight path may end, for example, on a steep cliff, i.e the algorithm finds a local optimum that is not identical with the global optimum. Evolutionary optimization methods are a way to explore the search space in a more random fashion - avoiding getting stuck in local optima and hopefully finding the local optimum.

BUT:

- There is no guarantee that these algorithms will succeed.
- The search can be very long-lasting.

AND:

- Modern Algorithms - as for example Random Forest or Gradient Boosting Trees - have randomness build in.

25.3 Random Forest

Random Forest is an example of classifier Bootstrap Aggregation or bagging.

- trees are not very deep (only stumps)
- each tree is build on a subsample of data and/or columns – choosen randomly
- results of the individual trees are aggregated (mean)

25.3.1 Pros of Random Forest:

- trees can be trained independently: easy to parallelize
- classification and regression possible
- all other pros of trees like: handling of missing values, insensitive to outliers, numerical and categorical data, etc..
- can give a variance estimate (confidence intervals): mean prediction and variance of prediction (see SMAC in AutoML)
- averaging allows for arbitrary non-linear relationships

25.3.2 Cons of Random Forest:

- black box algorithm: hard to interpret; (see feature importance)

```
<IPython.core.display.HTML object>
```

CHAPTER

TWENTYSIX

GRADIENT BOOSTED TREES

26.1 Explanation of Gradient Boosted Trees:

- we start with a prediction $f_0(x)$, i.e. the mean of y ($\bar{y}$) for regression or the most frequent class in case of classification
- the difference between the actual values y_i and our initial start value $f_0(x)$ (residuals) is to be predicted by the first tree T_1; the tree uses the variables $\boldsymbol{X}$ to find a rule to group similar residuals in common nodes.
- the new prediction of the tree T_1 is added to our initial start value and weighted by the learning parameter; now we get our prediction at iteration 1: $f_1(x) = f_0 + \alpha T_1(\boldsymbol{x}, y - f_0(x))$, i.e. we train a tree T_1 to correctly classify the residuals $y - f_0(x)$ with a rule induced on $\boldsymbol{x}$ – our variables. The result of this tree is weighted with leraning-rate α and added to the current estimate $f_0(x)$.
- this procedure is repeated until we can not find any more trees that substantially reduce our error or until the maximum number of iterations is reached

Since there are many different loss-functions possible for Gradient Boosting Trees (not only regression), we are looking for a more general formula that defines the best update to our current prediction made by the next tree.

26.2 most important parameters for stochastic gradient-boosting:

- **learning_rate**: the factor α in the above graphic
- **subsample**: takes part of the data without replacement; prevents overfitting
- **feature_fraction**: select a subset of the features for the next tree; prevents overfitting
- **num_leaves**: number of leaves; lightgbm fits level-wise and leaf-wise; prevents overfitting
- **max_depth**: number of levels to grow the tree; prevents overfitting
- **num_iterations**: number of trees to grow
- **max_bin**: in lightgbm all features are discretized by binning them; the number of bins for a feature is given by **max_bin** (this is what makes lightgbm super-fast)
- **lambda_l1/lambda_l2**: regularizes the leaf-weights; **excurs**: the result assigned to all cases that end up in one leaf is called the weight; for regression this is a continuous value and also for classification since the result is ultimately passed through a sigmoid-function that assigns then values between 0 and 1;

For a speed comparison between lightgbm and xgboost see e.g. results from 2017 Meanwhile xgboost catched up with lightgbm. A good paper, describing how the features of lightgbm have beed added to xgboost is this one. Less mathematical however, is this report here: https://everdark.github.io/k9/notebooks/ml/gradient_boosting/gbt.nb.html#5_lightgbm

1. Morning
   - Data Modelling & Cross Validation
   - data leakage & dependent data
   - imbalanced data (example in python)
   - study: Ebanking Fraud
   - Q&A
2. Afternoon
   - Data Basics & historical perspective
   - Linear Regression
   - Trees
   - house prices (regression example in python)
   - **Clustering**
   - bonus: Hyperparameter Optimization and AutoML
   - Q&A

CHAPTER

TWENTYSEVEN

CLUSTERING

Clustering is one of the hardest problems in ML, because:

- you do not know how many clusters there are
- it is unclear, which variables to use
- the scale of the variables is important for similarity

```
Automatically created module for IPython interactive environment
```

27.1 SOM

https://www.scribd.com/document/380399478/Teuvo-Kohonen-Self-Organizing-Maps

Also mention: Vector Quantization

Teuvo Kohonen, 1995: Self-Organizing Maps

Information Sciences
T. Kohonen
Self-Organizing Maps
Third Edition
Springer

Vapnik, 1992 SVM
SVM
Quinlan, 1986: ID3
Quinlan, 1993: C4.5
Friedman 1999 MART
Guolin Ke 2016 LightGBM
Tree Methods
Morgan & Sonquist, 1963: AID (automatic interaction detector)
Breiman et al. 1984: CART
Freud & Shapire: 1996 AdaBoost
Breiman et al. 2001: Random Forests
Chen, 2014 XGBoost
Sermanet, LeCun: Traffic sign recognition with multi-scale Convolutional Networks
Neural Networks
Hebb: 1949 'cells (neurons) that fire together wire together'
Rumelhart, Hinton, Williams: Learning representations by back-propagating errors
Hochreiter, Schmidhuber, 1997: Long Short-Term Memory
Kaggle: 2013 Cats vs. Dogs
ridge regression: Hoerl and Kennard, 1970
MARS multivariate adaptive regression splines: Friedman 1991
Linear Models (regression)
logistic regression: Walker and Duncan 1967
lasso: 1986, Tibshirani 1996
1940
1950
1960
1970
1980
1990
2000
2010
2020

Self-organizing map
Topic
Compare
Worldwide
2004 - present
All categories
Web Search
Interest over time
100
75
50
25
Note
Note
Jan 1, 2004
Jul 1, 2009
Jan 1, 2015
Jul 1, 2020

Cluster #2
Cluster #3
Cluster #1

umatrix
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1.0
0.8
0.6
0.4
0.2
0.0

```
([], [])
```

umatrix
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1.0
0.8
0.6
0.4
0.2
0.0

```
([], [])
```

umatrix
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1.0
0.8
0.6
0.4
0.2
0.0

```
([], [])
```

umatrix
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1.0
0.8
0.6
0.4
0.2
0.0

```
([], [])
```

umatrix
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1.0
0.8
0.6
0.4
0.2
0.0

```
([], [])
```

umatrix
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1.0
0.8
0.6
0.4
0.2
0.0

```
([], [])
```

27.2 Some considerations regarding the scale of used variables

In clustering we try to find observations that are near to each other and further away from dissimilar observations. The Notion of 'similar' only makes sense, when we pick out some variables that seem to be salient for the comparison of observations. Most often we use similarity measures as for example Euclidean Distance

$$D_{m,n} = \sqrt{\sum_{i=0}^{n} (m_i - n_i)^2} \tag{27.1}$$

Here, m and n are two observations with measures on n different variables. The Euclidean Distance is well known from the **Pythagorean theorem**: $a^2 + b^2 = c^2$; it is just its extensions to more than two dimensions.

What happens when the Variables are scaled differently? Consider for example: **absolute spending in different categories vs. relative (percentage of) spending in different categories**

absolute spendings:

```
<AxesSubplot:>
```

```
              a             b             c
a      0.000000  22219.023381  21100.947846
b  22219.023381      0.000000  28928.100525
c  21100.947846  28928.100525      0.000000
```

relative spendings:

```
<AxesSubplot:>
```

```
          a         b         c
a  0.000000  0.227983  0.252631
```

(continues on next page)

(continued from previous page)

```
b  0.227983  0.000000  0.410489
c  0.252631  0.410489  0.000000
```

As long as all variables have the same meaning and the same scale, clustering is still straight forward. But what are we going to do, if we have different variables with different scales? Consider adding the Age of the customers. A Difference in Age of 40 years adds not much to the distance of absolute spendings (in thousands of CHF), But will dominate the distance measure for relative spendings (percentages between 0 and 1).

27.3 Spectral clustering

Spectral clustering originates from Graph Theory (Spectral Analysis) The algorithms works as follows:

1. a connectivity matrix / affinity matrix W is formed. This matrix is symmetric and is n x n, where n is the number of observations. If there is a link between observation i and observation j there is a 1 in the matrix at positions W_{ij} and W_{ji}. Alternatively, for the affinity matrix, there is a similarity measure indicating the closeness of two observations.
2. the degree matrix D is formed. It is a diagonal matrix that contains for each observation the number of links (or sum of similarities) to other observations.

1. the laplacian matrix L is formed by $L = D - W$. This matrix has several interesting properties:

- the smallest eigenvalue is always 0. If all observations are connected (fully connected), there is exactly one eigenvalue that is 0.
- the corresponding eigenvector is constant.
- are there r connected components (cluster) in the data and if observations are ordered accordingly, the laplacian has block diagonal form:

$$\begin{bmatrix} L_1 & & \\ & \ddots & \\ & & L_r \end{bmatrix}$$

- the blocks L_i are proper laplacian matrices on their own. Since each component (cluster) is connected within, L_i has exactly one eigenvalue that is 0. The corresponding eigenvector is constant and is zero for all other components. These eigenvectors hence are indicator vectors for their component.

```
# this simple example is taken from https://towardsdatascience.com/unsupervised-
↪machine-learning-spectral-clustering-algorithm-implemented-from-scratch-in-python-
↪205c87271045
import numpy as np
import matplotlib.pyplot as plt
X = np.array([
    [1, 3], [2, 1], [1, 1],
    [3, 2], [7, 8], [9, 8],
    [9, 9], [8, 7], [13, 14],
    [14, 14], [15, 16], [14, 15]
])
plt.scatter(X[:,0], X[:,1], alpha=0.7, edgecolors='b')
plt.xlabel('Weight')
plt.ylabel('Height')
```

```
Text(0, 0.5, 'Height')
```

Next we buid the connectivity matrix W (also called adjacency matrix). We see, it has three connected components (clusters):

```
from sklearn.metrics import pairwise_distances
W = pairwise_distances(X, metric="euclidean")
vectorizer = np.vectorize(lambda x: 1 if x < 5 else 0)
W = np.vectorize(vectorizer)(W)
print(W)
```

```
[[1 1 1 1 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 0 1 1 1 1]]
```

Next we buid the degree matrix D and finally the laplacian as $L = D - W$

```
#D = np.diag(np.reshape(W.dot(np.ones((W.shape[1], 1))), (W.shape[1],)))
D = np.diag(W.sum(axis=1))
print(D)
```

```
[[4 0 0 0 0 0 0 0 0 0 0 0]
 [0 4 0 0 0 0 0 0 0 0 0 0]
 [0 0 4 0 0 0 0 0 0 0 0 0]
 [0 0 0 4 0 0 0 0 0 0 0 0]
 [0 0 0 0 4 0 0 0 0 0 0 0]
```

(continues on next page)

(continued from previous page)

```
 [0 0 0 0 0 4 0 0 0 0 0 0]
 [0 0 0 0 0 0 4 0 0 0 0 0]
 [0 0 0 0 0 0 0 4 0 0 0 0]
 [0 0 0 0 0 0 0 0 4 0 0 0]
 [0 0 0 0 0 0 0 0 0 4 0 0]
 [0 0 0 0 0 0 0 0 0 0 4 0]
 [0 0 0 0 0 0 0 0 0 0 0 4]]
```

```
L = D - W
print(L)
```

```
[[ 3 -1 -1 -1  0  0  0  0  0  0  0  0]
 [-1  3 -1 -1  0  0  0  0  0  0  0  0]
 [-1 -1  3 -1  0  0  0  0  0  0  0  0]
 [-1 -1 -1  3  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  3 -1 -1 -1  0  0  0  0]
 [ 0  0  0  0 -1  3 -1 -1  0  0  0  0]
 [ 0  0  0  0 -1 -1  3 -1  0  0  0  0]
 [ 0  0  0  0 -1 -1 -1  3  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  3 -1 -1 -1]
 [ 0  0  0  0  0  0  0  0 -1  3 -1 -1]
 [ 0  0  0  0  0  0  0  0 -1 -1  3 -1]
 [ 0  0  0  0  0  0  0  0 -1 -1 -1  3]]
```

```
# eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(L)

# sort these based on the eigenvalues
eigenvectors = eigenvectors[:,np.argsort(eigenvalues)]
eigenvalues = eigenvalues[np.argsort(eigenvalues)]
plt.scatter(np.arange(1, W.shape[1]+1), eigenvalues, alpha=0.7, edgecolors='b')
plt.xlabel('sorted eigenvalues')
plt.ylabel('eigenvalues')
```

```
Text(0, 0.5, 'eigenvalues')
```

```
print('corresponding eigenvectors')
print(eigenvectors[:, 0:3])
```

```
corresponding eigenvectors
[[-0.5  0.   0. ]
 [-0.5  0.   0. ]
 [-0.5  0.   0. ]
 [-0.5  0.   0. ]
 [ 0.  -0.5  0. ]
 [ 0.  -0.5  0. ]
 [ 0.  -0.5  0. ]
 [ 0.  -0.5  0. ]
 [ 0.   0.  -0.5]
 [ 0.   0.  -0.5]
 [ 0.   0.  -0.5]
 [ 0.   0.  -0.5]]
```

This is a somewhat contrieved example but it illustrates the principal idea. For normal data that is not as clear as in our example, the spectral embedding matrix (eigenvectors belonging to the s smallest eigenvalues) can be clustered by k-means. Another possibility is to start with the embedding matrix as a indicator matrix and optimize it as to maximize the in-component connectivity (association) and minimize the between component connectivity (cuts)(see https://www1.icsi.berkeley.edu/~stellayu/publication/doc/2003kwayICCV.pdf). In the python sklearn library the last method is called 'discretize'.

27.4 Example: 20 newsgroups

We are going to demonstrate the virtues of spectral clustering on a text-clustering example. The data is a common dataset with nearly balanced classes. Hence, k-means should also be appropriate.

Here is one example:

```
From: sigma@rahul.net (Kevin Martin)
Subject: Re: Stay Away from MAG Innovision!!!
Nntp-Posting-Host: bolero
Organization: a2i network
Lines: 10

In <16BB58B33.D1SAR@VM1.CC.UAKRON.EDU> D1SAR@VM1.CC.UAKRON.EDU (Steve Rimar) writes:
>My Mag MX15F works fine....................

Mine was beautiful for a year and a half.  Then it went <foomp>.  I bought
a ViewSonic 6FS instead.  Another great monitor, IMHO.

--
Kevin Martin
sigma@rahul.net
"I gotta get me another hat."
```

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

27.5 Organization

1. We build tf-idf vectors since we are following the bag-of-words approach
2. For building the adjacency/connectivity/affinity matrix we have several possibilities. We will discuss the most relevant ones
3. we compare different cluster solution and evaluate their results on the 20 different target labels of the newsgroup posts. For doing this, we introduce the the *adusted rand index*.
4. Since we are always curious we verify if we could have guessed the right number of clusters from the eigenvalue criterion as discussed above

```
#%reload_ext autoreload
#%autoreload 2
```

(continues on next page)

(continued from previous page)

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
from sklearn.cluster import SpectralClustering
from sklearn.cluster import KMeans
from sklearn import metrics
import nmslib
dataset = fetch_20newsgroups(subset='all', shuffle=True, download_if_missing=True)
# http://qwone.com/~jason/20Newsgroups/

np.random.seed(123)
texts = dataset.data # Extract text
target = dataset.target # Extract target
display(len(texts))
```

```
18846
```

CHAPTER

TWENTYEIGHT

TF-IDF

refresher:

$w_{i,j} = \text{tf}_{i,j} \cdot \log \frac{N}{\text{df}_i}$ with $\text{tf}_{i,j}$ is the frequency of term i in document j df_i is the number of documents containing term i

Since we want to cluster newsgroup posts we are more interested in words that are nearly unique to certain groups. By setting max_df = 0.3 we ensure that only words are considered that are not too common, i.e. only in 30% of all posts. By contrast, words that are very seldom but are concerned with special topics discussed in the groups are most important for our endeavor. Hence, there is no limit for min_df.We tell the vectorizer to remove common english stop words.By default the vectors returned are normed, i.e. they all have length 1. This is important when we compute the cosine similarity between the vectors.

28.1 Rand Index

The rand index is a measure of how well two partitions of a set of objects coincide:

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}},$$

where a is the number of pairs of elements that are in the same subset for both partitions b is the number of pairs of elements that are in different subsets for both partitions c is the number of pairs of elements that are in the same subset for the first partition but not for the second d is the number of pairs of elements that are in the different subsets for the first partition but in the same subset for the second partition

The Rand index is the percentage of consistent/congruent decisions for the two partitions.

```
# norm='l2' is default
vectorizer = TfidfVectorizer(stop_words='english', max_df = 0.3)
X = vectorizer.fit_transform(texts)

print(f'{X.shape[0]}, {X.shape[1]}')
```

```
18846, 173438
```

28.2 Connectivity / adjacency / affinity matrix

We have several possibilities to express the similarity / connectedness between observations. Some of common ones:

1. The ϵ neighborhood graph: all observations whose pairwise distance is less than ϵ are connected. (This was used in our contrieved example)
2. k-nearest neighborhood graph: for every observation, the k-nearest neighbors get connected. Since the nearest neighbors relationsship is not symmetric, we have to adjust the resulting adjacency matrix: $W_{symm} = 0.5 \cdot W + W^T$
3. fully connected graph: all observations are connected with weights equal to the respective similarity. Possible choices here are

- the gaussian kernel: $\exp^{-(x_i - x_j)^2/(2\sigma^2)}$; In sklearn the term $1/(2\sigma^2) = \gamma$; the gaussian kernel is also called radial basis function kernel (rbf) or euclidean kernel. For identical vectors $\exp^0 = 1$; For dissimilar vectors the term assymptotically approaches 0.
- the cosine kernel: $\frac{x_i \cdot x_j^T}{||x_i||||x_j||}$; here $||x_i||$ is the norm (length) of vector x_i, normalizing the vector to length 1. When the vectors are already normalized, the cosine similarity simplifies to $x_i \cdot x_j^T$. This is the linear kernel. The cosine measures the angle between two vectors. For identical vectors, the cosine is 1, for orthogonal vectors it is 0.

CHAPTER

TWENTYNINE

OUR CHOICE

Since we deal with very long and sparse vectors, euclidean distance is not a good choice. This has to do with the *curse of dimensionality* (search for it). Typically, for these long tf-idf vectors cosine similarity is used. For bigger data sets, I can not see any reason why we should compute the similarities for all points in the set. Probably, it is sufficient to consider only th k nearest neighbors of each point. This allows us to use sparse matrices, that are far more memory efficient. The sklearn spectral clustering implementation can deal with those matrices. As we will see, this simplification further reduces noise in the data and encourages better overall solutions.

CHAPTER

THIRTY

APPROXIMATE NEAREST NEIGHBORS

Computing all mutual distances between the 18846 posts can be very time and memory consuming. Theoretically, we only have to compute the triangular matrix minus the diagonal (distance matrices are symmetrical): $0.5 \cdot 18846^2 - 18846/2 = 177576435$ mutual distances. A clever idea here, is to compute only the k nearest neighbors of each post and treat all other posts as maximal distant. There are a lot of algorithms around for computing nearest neighbors, the most efficient relying on kd-trees. However, there are also approximate nearest neighbor algorithms that are nearly 100% accurate and are even faster than kd-trees. One of those is the hnsw-algorithm as implemented in the python nmslib (an api for the underlying c++ code). https://github.com/erikbern/ann-benchmarks

```
# see discussions about 'curse of dimensionality'; to be safe, we opt for cosine-
↪similarity
# since we want to be most efficient in everything we do, we use sparse matrices and␣
↪vectors
index = nmslib.init(method='hnsw', space='cosinesimil_sparse', data_type=nmslib.
↪DataType.SPARSE_VECTOR)
index.addDataPointBatch(X)
index_time_params = {'post':2}
index.createIndex(index_time_params, print_progress=True)

# now we can query the index
# By computing distances for the k=1000 nearest neighbors, we have to store 1000 *␣
↪18846 = 18846000
# distances; but compared to the triangular matrix approach discussed above, we still␣
↪save 158730435
# entries in our matrix.

nn = 1000
neighbors = index.knnQueryBatch(X, k=nn, num_threads=4)
```

```
neighbors[0:3]
```

```
[(array([    0, 18553, 11184, 16876, 11946, 16873, 11485,  2729, 11312,
          6760,  7949, 17776, 13476,  3309,  2867,  8311, 12784,  1154,
          7476,  7330,  8755, 13588, 13897, 17132,  6839,  3787,  3042,
           629, 15810, 18704,   333,  3925,  9968, 16990,  8496,  2361,
          6424, 15497,  9822, 11135, 13042, 18583,  5707, 15814,  7020,
          7142, 11961, 11029,  1803, 11246, 10832, 14097, 10344, 15032,
          7931, 13927,   458, 17853, 16067, 14836,  4506, 12084,  2853,
          4459, 15130, 17520, 16621, 10478, 13146, 13207,  8265, 15036,
         12512,  7803, 17900,  6566, 17040,  3349,     7, 13893,  2967,
         14123, 12769, 17020, 18590,  1898, 10391, 14628,  1469, 13250,
          1431, 16369,  3068, 10040, 11045, 10974, 15167,  9902,  6613,
           959], dtype=int32),
  array([0.        , 0.63908607, 0.669317  , 0.6755365 , 0.6795682 ,
```

(continues on next page)

(continued from previous page)

```
        0.6914151 , 0.70722437, 0.7155305 , 0.7231293 , 0.7415422 ,
        0.7510953 , 0.76321185, 0.7724261 , 0.7777292 , 0.77935755,
        0.7860545 , 0.8014517 , 0.8017465 , 0.80473745, 0.8059828 ,
        0.80765975, 0.8108567 , 0.8144285 , 0.81463706, 0.8146501 ,
        0.81604654, 0.81780756, 0.81853294, 0.8226091 , 0.8253168 ,
        0.8269756 , 0.8272094 , 0.82753253, 0.832714  , 0.83565545,
        0.83763695, 0.84240353, 0.8450652 , 0.8454548 , 0.84554344,
        0.8469504 , 0.84886557, 0.85017437, 0.8515262 , 0.8534748 ,
        0.8535583 , 0.8540908 , 0.8569978 , 0.8593518 , 0.86037207,
        0.8613587 , 0.8628519 , 0.8643788 , 0.8650072 , 0.8686269 ,
        0.8700882 , 0.8730786 , 0.8753253 , 0.87538624, 0.87732553,
        0.8785739 , 0.8803475 , 0.8814558 , 0.8818229 , 0.8836487 ,
        0.8837902 , 0.88500327, 0.8850998 , 0.88521737, 0.8865601 ,
        0.88761014, 0.8889066 , 0.8891484 , 0.8891751 , 0.8892081 ,
        0.8903198 , 0.89066046, 0.89234483, 0.89260024, 0.89307475,
        0.89561146, 0.89589816, 0.89832234, 0.89868456, 0.8989996 ,
        0.8998436 , 0.90027505, 0.9010556 , 0.9020781 , 0.9055549 ,
        0.90569705, 0.90630126, 0.90735704, 0.9079704 , 0.9080308 ,
        0.9089797 , 0.909623  , 0.9108277 , 0.9108595 , 0.9108808 ],
       dtype=float32)),
(array([    1, 11766,  7423,  6468,  6102, 10510, 16899,  1587,  8028,
         8005,  7609,  4800, 17882, 15953,  4582, 12907,  7943, 14461,
         1578,  4132,   437,  4277,  9712,  1425,  4904,  1674, 13677,
         4301, 13416,   777, 12782, 13463, 14806, 13213,  6574, 13653,
         1140, 12095,   820,  2783, 10416,  5639, 10442,  8486,  7638,
        18446,  3751,   845, 17619,  1648,  8586,  8772, 18823,   134,
         6115, 11394, 13339, 17526, 16883,   875,  4739,   738, 13985,
         8460,  6427, 17228,  6889,  4363, 14301,  8563,  1457, 12688,
        15129, 17449,  9731, 12413,  7858, 14259,   253,  9652, 15670,
         8764, 17087,  9998,  6638,  2276,  7980, 17831, 17765, 17358,
         1175, 16537, 16167,  7780, 17516, 14695,  2838,  1503,  4027,
         5806], dtype=int32),
 array([0.        , 0.6764641 , 0.74253815, 0.7553041 , 0.76306784,
        0.7711086 , 0.7799365 , 0.780106  , 0.78269386, 0.7828499 ,
        0.7828758 , 0.784539  , 0.78670365, 0.7882724 , 0.7885906 ,
        0.797265  , 0.80187076, 0.802405  , 0.8029315 , 0.80353254,
        0.80902755, 0.8120299 , 0.8170264 , 0.81749696, 0.8178127 ,
        0.8206302 , 0.8228782 , 0.8252084 , 0.826402  , 0.82888323,
        0.82903314, 0.8292601 , 0.82927394, 0.8304367 , 0.8350926 ,
        0.8382005 , 0.8384994 , 0.84133023, 0.8426109 , 0.84430575,
        0.84561086, 0.84563506, 0.84599364, 0.8468888 , 0.8490473 ,
        0.8501638 , 0.85109127, 0.851439  , 0.85309803, 0.8534955 ,
        0.85463166, 0.8546634 , 0.856802  , 0.85953075, 0.8595509 ,
        0.8596823 , 0.8602279 , 0.861897  , 0.86335284, 0.8658271 ,
        0.8666834 , 0.8676325 , 0.86765236, 0.8682556 , 0.86828136,
        0.86998975, 0.87009454, 0.8708112 , 0.87231827, 0.87314016,
        0.8755579 , 0.87881505, 0.8794431 , 0.8799997 , 0.88102365,
        0.88231266, 0.88626516, 0.8864002 , 0.8870982 , 0.8873377 ,
        0.88741034, 0.88744533, 0.8879509 , 0.89046234, 0.8919709 ,
        0.89297944, 0.8942795 , 0.89578915, 0.89626247, 0.8970282 ,
        0.89725167, 0.89876753, 0.8994845 , 0.900252  , 0.90164286,
        0.90257215, 0.9031204 , 0.9032326 , 0.90526605, 0.9073184 ],
       dtype=float32)),
(array([    2, 12577,  5478, 15736, 10818,  3973,  1382, 16593,  4949,
        16086,  7747,  7586,   115,   743, 11505,  2070,  6514, 14587,
         9424,  5761, 13039,  2914, 11949,  3991, 10796, 10800, 12879,
        13621,  1791,  3058,  3794,  6028,  2262, 11165, 14922,  8374,
```

(continues on next page)

(continued from previous page)

```
          6057,  1229, 13813, 10241,   991,  2191, 14417,   114, 13170,
         18345, 11710, 15688,  9075, 15820, 17789, 14095, 10126,  1784,
          9368,  2596,  6639,  7615, 10537,  2603, 17523, 18486,   196,
         13365, 16950,  2754,  5598,  8372,  5777,  9200,  2008, 18006,
         17137,   731,  1236,  5967,  7182,  9658,  2427,  6958, 17731,
          5630, 13576, 11702,   858, 15778, 17675, 17458,  2978,  8354,
          5175,  9232,   736, 14002,  5695, 13958, 13424,   552,   521,
          6149], dtype=int32),
  array([0.        , 0.31400347, 0.395284  , 0.40862632, 0.48660403,
         0.6930011 , 0.70131546, 0.7452333 , 0.75990844, 0.76281995,
         0.76501834, 0.76625437, 0.77072847, 0.7735661 , 0.77506244,
         0.77915657, 0.7807083 , 0.78274506, 0.78300846, 0.7831948 ,
         0.78646934, 0.79291785, 0.79638773, 0.8006463 , 0.8021226 ,
         0.80240417, 0.8031671 , 0.8063524 , 0.8082599 , 0.8088174 ,
         0.8091037 , 0.8097666 , 0.810273  , 0.81131786, 0.812379  ,
         0.8135287 , 0.81699   , 0.82013595, 0.8219625 , 0.8223238 ,
         0.8247599 , 0.8261044 , 0.8263263 , 0.82775855, 0.8294813 ,
         0.83061796, 0.8330438 , 0.834819  , 0.83547986, 0.8369826 ,
         0.8370266 , 0.83754146, 0.83761626, 0.83841634, 0.8390262 ,
         0.84067756, 0.84150666, 0.8430389 , 0.8437043 , 0.8454485 ,
         0.8459446 , 0.84668833, 0.84833264, 0.8484504 , 0.8486481 ,
         0.85070646, 0.85274833, 0.85281026, 0.8543711 , 0.85746324,
         0.85863894, 0.86052996, 0.8620279 , 0.8627942 , 0.8636267 ,
         0.8681772 , 0.8683529 , 0.8684674 , 0.86972046, 0.8704043 ,
         0.8710359 , 0.87433565, 0.8752069 , 0.8757434 , 0.87733775,
         0.87839425, 0.8785302 , 0.87890065, 0.8792665 , 0.88187623,
         0.88302463, 0.8832308 , 0.8846249 , 0.88844943, 0.88874626,
         0.88897663, 0.8893845 , 0.8897048 , 0.8897395 , 0.8898656 ],
        dtype=float32))]
```

Next, we build our affinity matrix. Sparse matrices only store the indices and the data for entries in the matrix that acutally contain data.

```
from scipy.sparse import csc_matrix
# next we construct a sparse matrix with the distances as measured by cosine␣
↪similarity
col = np.array([i for n in neighbors for i in n[0].tolist()]) # column indices of␣
↪nearest neighbors
row = np.repeat(np.arange(0, len(neighbors)), np.array([len(n[0]) for n in␣
↪neighbors])) # the row index for each nearest neighbor
# data = np.array([i for n in neighbors for i in n[1].tolist()]) # the similarities
data = np.array([1 for n in neighbors for i in n[1].tolist()]) # the similarities

# btw, if you do not understand what's going on in the list part of the construction␣
↪of the col and
# the data variable above, I stronly recommend to have a look at *list comprehension*;
↪
# list comprehension is a super fast way to get things done in python

affinity = csc_matrix((data, (row, col)), shape = (X.shape[0], X.shape[0]))
```

```
# n_clusters = 20 because we know, that there are 20 different newsgroups
# n_components = 21; this is cheated because it should normally be set to the same␣
↪number as
#                the n_clusters, but 21 yielded a better solution
# assign_labels = 'discretize' defines how the clusters are found within the
```

(continues on next page)

(continued from previous page)

```
#                  embedding matrix (matrix composed of eigenvectors belonging to the␣
 ↪n_components
#                  smallest eigenvalues). As discussed above, 'discretize' maximizes␣
 ↪the
#                  in-component connectivity and minimizes the between-component␣
 ↪connectivity
# affinity = 'precomputed' indicates that the affinity matrix is provided by the user
# eigen_solver = 'amg' assumes the python-package pyamg is installed. It is by far␣
 ↪the fastest
#                 way the eigenvalue decomposition
#
solution = SpectralClustering(n_clusters = 20, n_components = 21, assign_labels=
 ↪'discretize',\
                              affinity = 'precomputed',\
                              eigen_solver='amg').fit(0.5 * (affinity + affinity.T))
metrics.adjusted_rand_score(solution.labels_, target)
```

```
0.3312465898113292
```

So, let's see what our nearest-neighbor trick was good for. In the next cell we compute the spectral clustering for the whole tf-idf matrix and leave it to sklearn to compute all cosine similarities. As discussed above, since the tf-idf vectors are by default normalized, we can use the 'linear' kernel instead of the 'cosine' kernel. This is faster because the additional steps of getting the norms and dividing is not needed.

```
# linear kernel instead of cosine_similarity if tf-idf vectors are already normalized
solution = SpectralClustering(n_clusters=20, n_components = 21, assign_labels=
 ↪'discretize',\
                              affinity='linear',eigen_solver='amg').fit(X)
metrics.adjusted_rand_score(solution.labels_, target)
```

```
0.275483821009973
```

What would the solution be like if we would have used ordinary euclidean distance (=rbf kernel=gaussian kernel)? gamma ($\gamma = \frac{1}{2\sigma^2}$) defines the size of the neighborhood and is similar to the choice of the parameter k in the approx. k-nearest neighbor algorithm we used in the construction of our affinity matrix.

```
solution = SpectralClustering(n_clusters=20, n_components = 21, assign_labels=
 ↪'discretize',\
                              affinity='rbf',eigen_solver='amg',\
                              gamma = 0.7).fit(X)
metrics.adjusted_rand_score(solution.labels_, target)
```

```
0.17161380707083057
```

How is ordinary k-means performing? Clustering is done on the tf-idf vectors.

```
solutionKMeans = KMeans(n_clusters=20, init='k-means++', max_iter=100, n_init=1).
 ↪fit(X)
metrics.adjusted_rand_score(solutionKMeans.labels_, target)
```

```
0.1760642244041933
```

Could we have guessed the right number of clusters from the eigenvalues of the laplacian as suggested in all tutorials about spectral clustering?

```
from scipy.sparse.csgraph import laplacian as csgraph_laplacian
from sklearn.utils import check_array
from sklearn.utils import check_random_state
from sklearn.utils.fixes import lobpcg
from sklearn.manifold._spectral_embedding import _set_diag
from scipy import sparse
from pyamg import smoothed_aggregation_solver
import matplotlib.pyplot as plt
```

```
norm_laplacian = True
random_state = check_random_state(1234)

connectivity =0.5 * (affinity + affinity.T)
laplacian, dd = csgraph_laplacian(connectivity, normed=norm_laplacian, return_
 ↪diag=True)
laplacian = check_array(laplacian, dtype=np.float64, accept_sparse=True)

laplacian = _set_diag(laplacian, 1, norm_laplacian)
diag_shift = 1e-5 * sparse.eye(laplacian.shape[0])
laplacian += diag_shift
ml = smoothed_aggregation_solver(check_array(laplacian, 'csr'))
laplacian -= diag_shift
n_components = laplacian.shape[0]
M = ml.aspreconditioner()
X = random_state.rand(laplacian.shape[0], n_components + 1)
X[:, 0] = dd.ravel()
eigs, diffusion_map = lobpcg(laplacian, X, M=M, tol=1.e-5, largest=False)
plt.scatter(np.arange(len(eigs)), eigs)
plt.grid()
plt.show()
```

```
plt.scatter(np.arange(len(eigs[0:255])), eigs[0:255])
plt.plot([19]* 2, [eigs[19]] * 2, 'ro')
plt.grid()
```

CHAPTER

THIRTYONE

DBSCAN

Density-based clustering of applications with noise.

DBSCAN has the concept of **core point**, **border point** and **outlier**. A data point is considered a core point if there are sufficient (***minPts***) data points in its vicinity of radius ***eps***. Border points are those points that are connected to core points but themselves have not enough data point in their neighborhood. Outliers are the points that are not connected to a core points.

The image is taken from wikipedia

- red points (e.g. point A) are **core points**
- yellow points (e.g. points B, c) are **border points**
- the blue point (N) is an **outlier**

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.5, min_samples=14, metric='cosine', n_jobs=-1)
clusters = db.fit(X)
```

```
np.unique(clusters.labels_)
```

```
array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
       16, 17, 18, 19, 20])
```

```
metrics.adjusted_rand_score(clusters.labels_, target)
```

```
8.75031255964891e-05
```

CHAPTER

THIRTYTWO

SOM

```
!pip install minisom
```

```
Collecting minisom
  Downloading MiniSom-2.3.0.tar.gz (8.8 kB)
Building wheels for collected packages: minisom
  Building wheel for minisom (setup.py) ... ?25ldone
?25h  Created wheel for minisom: filename=MiniSom-2.3.0-py3-none-any.whl size=9018␣
↪sha256=82e6f8beb422ec311bb851ae0baca7c5bb967a356f0f79e354173f732e5ee698
  Stored in directory: /home/martin/.cache/pip/wheels/d4/ca/4a/
↪488772b0399fec45ff53132ed14c948dec4b30deee3a532f80
Successfully built minisom
Installing collected packages: minisom
Successfully installed minisom-2.3.0
```

```
from minisom import MiniSom
som = MiniSom(x=20, y=20, input_len=X.shape[1], sigma=1, learning_rate=0.5)
```

```
som.random_weights_init(X)
som.train_random(data=X, num_iteration=100)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-87-bdd4d7e5b832> in <module>
----> 1 som.random_weights_init(X)
      2 som.train_random(data=X, num_iteration=100)

~/miniconda3/envs/imbalanced/lib/python3.7/site-packages/minisom.py in random_weights_
↪init(self, data)
    349         """Initializes the weights of the SOM
    350         picking random samples from data."""
--> 351         self._check_input_len(data)
    352         it = nditer(self._activation_map, flags=['multi_index'])
    353         while not it.finished:

~/miniconda3/envs/imbalanced/lib/python3.7/site-packages/minisom.py in _check_input_
↪len(self, data)
    304     def _check_input_len(self, data):
    305         """Checks that the data in input is of the correct shape."""
--> 306         data_len = len(data[0])
    307         if self._input_len != data_len:
    308             msg = 'Received %d features, expected %d.' % (data_len,
```

(continues on next page)

(continued from previous page)

```
~/miniconda3/envs/imbalanced/lib/python3.7/site-packages/scipy/sparse/base.py in __
↪len__(self)
    289     # non-zeros is more important.  For now, raise an exception!
    290     def __len__(self):
--> 291         raise TypeError("sparse matrix length is ambiguous; use getnnz()"
    292                         " or shape[0]")
    293

TypeError: sparse matrix length is ambiguous; use getnnz() or shape[0]
```

```
X.shape
```

```
(18846, 173438)
```