DS-210: Programming for Data Science

Lecture 4: Sample predictive data analysis pipeline

Typical steps in predictive data analysis

- Collect, validate, and clean data
- Feature selection and additional data preparation
- Split data into training and testing set
- Train your algorithm
- Estimate its accuracy

Typical steps in predictive data analysis

- Collect, validate, and clean data
- Feature selection and additional data preparation
- Split data into training and testing set
- Train your algorithm
- Estimate its accuracy

In [1]:
```
# set up our environment
import pandas as pd

def print_file(filename):
    with open(filename) as f:
        print(f.read(),end='')
```

Data preparation

Data collection

- can be non-trivial
- may require combining information from several sources

Data validation

- ensure the proper format
- no missing values of any kind
- basic data consistency

Data preparation

Data collection

- can be non-trivial
- may require combining information from several sources

Data validation

- ensure the proper format
- no missing values of any kind
- basic data consistency

Data cleaning

- decide how to fix data
- examples:
  - remove duplicate entries?
  - drop entries with incorrect or missing info?
  - fix entries with incorrect or missing info?
  - fix misspellings?

Sample of data validation

In [2]:
```
filename = 'data_validation.csv'
print_file(filename)
```

```
Name;FavoriteNumber;PPG
Alice;7;13.2
;8;12.7
Carol;13;8.2
Dave;"five"
Jack;32;
;;
```

Sample of data validation

In [2]:
```
filename = 'data_validation.csv'
print_file(filename)
```

```
Name;FavoriteNumber;PPG
Alice;7;13.2
;8;12.7
Carol;13;8.2
Dave;"five"
Jack;32;
;;
```

In [3]:
```
data = pd.read_csv(filename, sep=';')
data
```

Out[3]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	NaN	8	12.7
2	Carol	13	8.2
3	Dave	five	NaN
4	Jack	32	NaN
5	NaN	NaN	NaN

Sample of data validation

In [2]:
```
filename = 'data_validation.csv'
print_file(filename)
```

```
Name;FavoriteNumber;PPG
Alice;7;13.2
;8;12.7
Carol;13;8.2
Dave;"five"
Jack;32;
;;
```

In [3]:
```
data = pd.read_csv(filename, sep=';')
data
```

Out[3]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	NaN	8	12.7
2	Carol	13	8.2
3	Dave	five	NaN
4	Jack	32	NaN
5	NaN	NaN	NaN

In [4]:
```
# check if types are as expected
# if not, incorrect entry
data.dtypes
```

Out[4]:
```
Name              object
FavoriteNumber    object
PPG              float64
dtype: object
```

Sample of data validation

In [2]:
```
filename = 'data_validation.csv'
print_file(filename)
```

```
Name;FavoriteNumber;PPG
Alice;7;13.2
;8;12.7
Carol;13;8.2
Dave;"five"
Jack;32;
;;
```

In [3]:
```
data = pd.read_csv(filename, sep=';')
data
```

Out[3]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	NaN	8	12.7
2	Carol	13	8.2
3	Dave	five	NaN
4	Jack	32	NaN
5	NaN	NaN	NaN

In [4]:
```
# check if types are as expected
# if not, incorrect entry
data.dtypes
```

Out[4]:
```
Name              object
FavoriteNumber    object
PPG              float64
dtype: object
```

In [5]:
```
# count missing entries in a column
column = 'PPG'
missing = pd.isnull(data[column]).sum()
print(f'{missing} missing entries in {column}')
```

```
3 missing entries in PPG
```

Sample of data cleaning

In [7]:
```
# drop entries with a NaN
data = pd.read_csv(filename, sep=';')
data = data.dropna(axis=0,how='any')
#data = data.dropna(axis=0,how='all')
data
```

Out[7]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
2	Carol	13	8.2

Sample of data cleaning

In [8]:
```
# drop entries with a NaN
data = pd.read_csv(filename, sep=';')
#data = data.dropna(axis=0,how='any')
data = data.dropna(axis=0,how='all')
data
```

Out[8]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	NaN	8	12.7
2	Carol	13	8.2
3	Dave	five	NaN
4	Jack	32	NaN

Sample of data cleaning

In [8]:
```
# drop entries with a NaN
data = pd.read_csv(filename, sep=';')
#data = data.dropna(axis=0,how='any')
data = data.dropna(axis=0,how='all')
data
```

Out[8]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	NaN	8	12.7
2	Carol	13	8.2
3	Dave	five	NaN
4	Jack	32	NaN

In [9]:
```
# replace NaNs with specific value
# data.fillna('????')
data['PPG'] = data['PPG'].fillna(0)
data['Name'] = data['Name'].fillna('????')
data['FavoriteNumber'] = data['FavoriteNumber']\
                            .fillna(0)
data
```

Out[9]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	????	8	12.7
2	Carol	13	8.2
3	Dave	five	0.0
4	Jack	32	0.0

Sample of data cleaning

In [8]:
```
# drop entries with a NaN
data = pd.read_csv(filename, sep=';')
#data = data.dropna(axis=0,how='any')
data = data.dropna(axis=0,how='all')
data
```

Out[8]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	NaN	8	12.7
2	Carol	13	8.2
3	Dave	five	NaN
4	Jack	32	NaN

In [9]:
```
# replace NaNs with specific value
# data.fillna('????')
data['PPG'] = data['PPG'].fillna(0)
data['Name'] = data['Name'].fillna('????')
data['FavoriteNumber'] = data['FavoriteNumber']\
                            .fillna(0)
data
```

Out[9]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	????	8	12.7
2	Carol	13	8.2
3	Dave	five	0.0
4	Jack	32	0.0

In [10]:
```
def fix(x):
    if x == "five":
        return 5
    return x

data['FavoriteNumber'] = data['FavoriteNumber']\
                            .apply(fix)
```

Sample of data cleaning

In [8]:
```
# drop entries with a NaN
data = pd.read_csv(filename, sep=';')
#data = data.dropna(axis=0,how='any')
data = data.dropna(axis=0,how='all')
data
```

Out[8]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	NaN	8	12.7
2	Carol	13	8.2
3	Dave	five	NaN
4	Jack	32	NaN

In [9]:
```
# replace NaNs with specific value
# data.fillna('????')
data['PPG'] = data['PPG'].fillna(0)
data['Name'] = data['Name'].fillna('????')
data['FavoriteNumber'] = data['FavoriteNumber']\
                            .fillna(0)
data
```

Out[9]:

	Name	FavoriteNumber	PPG
0	Alice	7	13.2
1	????	8	12.7
2	Carol	13	8.2
3	Dave	five	0.0
4	Jack	32	0.0

In [10]:
```
def fix(x):
    if x == "five":
        return 5
    return x

data['FavoriteNumber'] = data['FavoriteNumber']\
                            .apply(fix)
```

In [11]:
```
better_file = "cleaned.csv"
data.to_csv(better_file, sep=";")
print_file(better_file)
```

```
;Name;FavoriteNumber;PPG
0;Alice;7;13.2
1;????;8;12.7
2;Carol;13;8.2
3;Dave;5;0.0
4;Jack;32;0.0
```

Digression: Conditional expressions and lambda functions

How to create and pass function `fix(x)` more concisely:

```
def fix(x):
    if x == "five":
        return 5
    return x
```

Digression: Conditional expressions and lambda functions

How to create and pass function `fix(x)` more concisely:

```
def fix(x):
    if x == "five":
        return 5
    return x
```

Conditional expression: `<x> if <condition> else <y>`
(in C++: `<condition> ? <x> : <y>`)

```
In [12]: "all good" if 2 + 2 == 4 else "what??!!"

Out[12]: 'all good'
```

Digression: Conditional expressions and lambda functions

How to create and pass function `fix(x)` more concisely:

```
def fix(x):
    if x == "five":
        return 5
    return x
```

Conditional expression: `<x> if <condition> else <y>`
(in C++: `<condition> ? <x> : <y>`)

```
In [13]: "all good" if 2 + 2 == 5 else "what??!!"

Out[13]: 'what??!!'
```

Digression: Conditional expressions and lambda functions

How to create and pass function `fix(x)` more concisely:

```
def fix(x):
    if x == "five":
        return 5
    return x
```

Conditional expression: `<x> if <condition> else <y>`
(in C++: `<condition> ? <x> : <y>`)

```
In [13]: "all good" if 2 + 2 == 5 else "what??!!"

Out[13]: 'what??!!'
```

More concise version:

```
def fix(x):
    return 5 if x == "five" else x
```

Digression: Conditional expressions and lambda functions

Lambda functions: `lambda <x> : <expression>`

(in OCaml: `fun <x> -> <expression>`)
(in C++: `[] (<type-of-x> <x>) {return <expression;}`)

In [14]:
```
f = lambda x : x * x
f(16)
```

Out[14]: 256

Digression: Conditional expressions and lambda functions

Lambda functions: `lambda <x> : <expression>`

(in OCaml: `fun <x> -> <expression>`)
(in C++: `[] (<type-of-x> <x>) {return <expression;}`)

```
In [14]: f = lambda x : x * x
         f(16)

Out[14]: 256
```

OCaml: `fun x -> x * x`
C++: `[](int x) { return x * x;}`

Digression: Conditional expressions and lambda functions

Lambda functions: `lambda <x> : <expression>`

(in OCaml: `fun <x> -> <expression>`)
(in C++: `[] (<type-of-x> <x>) {return <expression;}`)

```
In [14]: f = lambda x : x * x
         f(16)

Out[14]: 256
```

OCaml: `fun x -> x * x`
C++: `[](int x) { return x * x;}`

We don't have to create a named function to execute this step

```
In [15]: data['FavoriteNumber'].apply(lambda x : 5 if x == "five" else x)

Out[15]: 0     7
         1     8
         2    13
         3     5
         4    32
         Name: FavoriteNumber, dtype: object
```

Feature selection and additional data preparation

Important: select a subset of available attributes, especially if you have few labeled samples

Otherwise: the training algorithm could focus on non-useful data

Feature selection and additional data preparation

Important: select a subset of available attributes, especially if you have few labeled samples

Otherwise: the training algorithm could focus on non-useful data

Additional data preparation (algorithm dependent)

- Normalize the selected features

Feature selection and additional data preparation

Important: select a subset of available attributes, especially if you have few labeled samples

Otherwise: the training algorithm could focus on non-useful data

Additional data preparation (algorithm dependent)

- Normalize the selected features

- Warning for decision trees in `scikit-learn` (applies to Homework 1):
  - The algorithm only works for numerical attributes
  - So you have to convert your data to numerical
  - For instance, replace `"Male"`/`"Female"` with `0`/`1`

Validation of results

General idea:

- split your data at random:
    - training set
    - test set
- use the training set to train your prediction model
- use the test set to see how well it performs

Validation of results

General idea:

- split your data at random:
  - training set
  - test set
- use the training set to train your prediction model
- use the test set to see how well it performs

```
In [16]: # split the data set
         from sklearn.datasets import load_iris
         from sklearn.model_selection import train_test_split
         from sklearn import tree
         iris = load_iris()
         X,y = iris.data,iris.target
         X_train,X_test,y_train,y_test = train_test_split(X,y)
```

Validation of results

General idea:

- split your data at random:
  - training set
  - test set
- use the training set to train your prediction model
- use the test set to see how well it performs

In [16]:
```
# split the data set
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import tree
iris = load_iris()
X,y = iris.data,iris.target
X_train,X_test,y_train,y_test = train_test_split(X,y)
```

In [17]:
```
# train your decision tree
clf = tree.DecisionTreeClassifier(max_leaf_nodes=3)
clf = clf.fit(X_train,y_train)
```

Validation of results

General idea:

- split your data at random:
  - training set
  - test set
- use the training set to train your prediction model
- use the test set to see how well it performs

In [16]:
```
# split the data set
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import tree
iris = load_iris()
X,y = iris.data,iris.target
X_train,X_test,y_train,y_test = train_test_split(X,y)
```

In [17]:
```
# train your decision tree
clf = tree.DecisionTreeClassifier(max_leaf_nodes=3)
clf = clf.fit(X_train,y_train)
```

In [18]:
```
# what was the prediction accuracy?
prediction = clf.predict(X_test)
correct = 0
for i in range(len(y_test)):
    if prediction[i] == y_test[i]:
        correct += 1
correct / len(y_test)
```

Out[18]: 0.9210526315789473