Data Mining in Python

Witek ten Hove

2022-10-24

Table of contents

Preface

Figure 1: CRISP-DM Model taken from: https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png

Data mining is the process of sorting through large datasets to identify patterns or relationships to inform business decisions. It is a crucial aspect of modern data analytics, particularly for industries that rely heavily on large amounts of data to inform their business operations.

Prerequisites

Before starting this module make sure you have:

- access to the book *Nield, T. (2022). Essential Math for Data Science. O'Reilly Media, Inc.*
- a data science environment setup

Purpose of this course

The general learning outcome of this course is:

> The student is able to perform a well-defined task independently in a relatively clearly arranged situation, or is able to perform in a complex and unpredictable situation under supervision.

The course will provide you with a few essential data mining skills. The focus will lie on non-linear modeling techniques - k-Nearest Neighbors (kNN) and Naive Bayes classification.

After a successful completion of the course, a student can demonstrate his or her ability to:

- explore and prepare data for a given non-linear model
- train en test a non-linear model
- evaluate the quality of a trained model

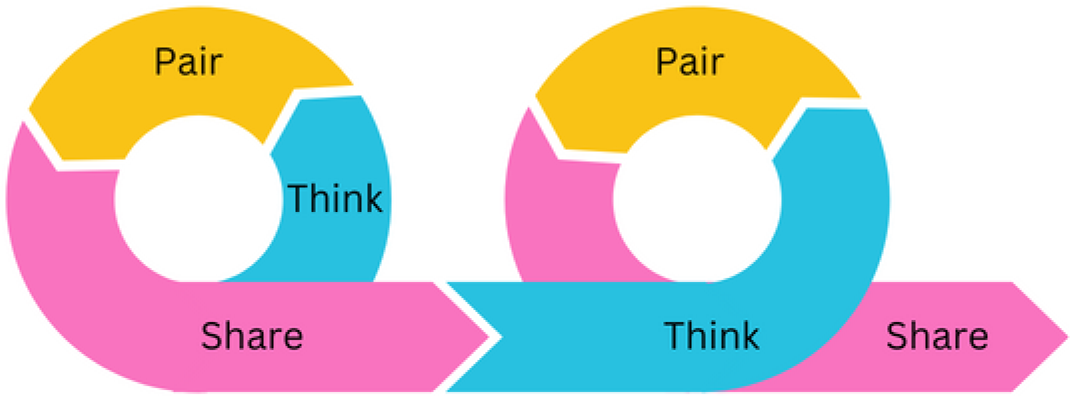
Structure of the course

Table 1: Course overview

Week nr.	Module name	Readings
2	Onboarding and Data Exploration	
3-4	Lazy Learning with kNN	Nield Ch.1 up to and including 'Exponents'
5-6	Probabilistic Learning with Naive Bayes Classification	Nield Ch.2 up to and including 'Probablity Math', Ch.3, Ch.4 up to and including 'What Is a Vector?'
7	Project Application	

Through the whole of the program you'll be working on your own data mining projects:

- You will setup your own data science environment
- Find and choose datasets for your projects
- Run several full data mining cycles
- Document and share your learnings
- Demonstrate you newly acquired competences and skills



Make sure all steps in the data mining process are properly documented. The quality of documentation must be such that an informed data specialist must be able to understand the challenge and the conclusions, the design decisions and the reasons for the choices made during the process.

- Stretch and Challenge: Advanced students can further research and explore new algorithms for data mining, comparing their performance with KNN and Naive Bayes.
- Inclusion: Students who are struggling can work with a partner or teacher during activities to ensure they comprehend the material.

Essential Math

For k-Nearest Neighbors

An essential element of the k-Nearest Neighbor model is *distance*. Several methods exist to calculate the distance between two points. One is the Euclidean distance. Let point p have

Cartesian coordinates (p_1, p_2) and let point q have coordinates (q_1, q_2). Then the distance between p and q is given by:

$$d(p,q) = \sqrt{\sum_{i=1}^{2} (p_i - q_i)^2}$$

For higher dimensions n this becomes:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Important math topics:

- Order of operation: deduct or square first?
- Variables and types: what are the variables in the above formulas and of what type are they?
- Functions: which are the dependent and which the independent variables?
- Summations: what is the value of $\sum_{i=3}^{4} (i^2)$
- Exponents: what is the value of $(\sum_{i=3}^{4} (i^2))^{-\frac{1}{2}}$

For Naive Bayes

- Probability math
- Descriptive statistics
- Vectors

About the author

Witek ten Hove is a senior instructor and researcher at HAN University of Applied Sciences. His main areas of expertise are Data en Web Technologies.

Through his extensive business experience in Finance and International Trade and thorough knowledge of modern data technologies, he is able to make connections between technology and business. As an open source evangelist he firmly believe in the power of knowledge sharing. His mission is to inspire business professionals and help them exploit the full potential of smart technologies.

He is the owner of Ten Hove Business Data Solutions, a consultancy and training company helping organizations to achieve maximum business value through data driven solutions.

1 Setting up your data science environment

Here's a general set of instructions for setting up a development environment that includes GitHub, Anaconda, and an Integrated Development Environment (IDE):

1. First, you'll need to install Git on your computer. Git is a version control system that allows you to track changes in your code and collaborate with other developers. You can download the latest version of Git from the official website: **https://git-scm.com/downloads**
2. Next, create a GitHub account if you don't already have one. GitHub is a web-based platform for version control and collaboration that uses Git. You can sign up for a free account at **https://github.com/**.
3. Anaconda is a distribution of Python and R that makes it easy to manage dependencies and packages for data science. You can download the latest version of Anaconda from the official website: **https://www.anaconda.com/products/distribution**.
4. After installing Anaconda, you can create a new environment for your data science project by opening Anaconda Navigator, then click on the Environments tab, and then click on the create button. You can then set the name of the environment, and the version of Python or R you want to use.
5. Finally, you can install your preferred IDE:
    1. Spyder IDE is included in your Anaconda installation. You might want to add the **Notebook plugin**.
    2. **Visual Studio code** with appropriate **extensions**.
    3. Rstudio can be downloaded from **https://rstudio.com/products/rstudio/download/#downlo**

Below you will find more detailed video instructions on installing and using the different tools in your development environment.

1.1 Working with Git and Github

1.2 Using Python virtual environments

1.3 Visual Studio Code

1.4 Working with Quarto

2 Data Understanding

Links:

1. www.kaggle.com/
2. datasetsearch.research.google…
3. data.fivethirtyeight.com/
4. data.gov/
5. github.com/search?q=dataset
6. data.nasa.gov/
7. selected datasets

Table 2.1: Lesson outline

	Topic	Tasks	Activities	Student	Teacher
1	Finding Data	Explore the different sources of data that may be used in data mining, and how to extract and access this data.	Think-Pair-Share: students will individually brainstorm potential sources of data, pair up with a partner to discuss, and then share with the class.	'We learned about various data sources and perspectives of different students during the brainstorming activity.'	'Our objective here is to generate a list of possible sources of data that we can use for data mining. As a teacher, I want you to participate actively in brainstorming and support each other's thoughts. As students, you will be able to collaborate and gain insights from your peers.'

	Topic	Tasks	Activities	Student	Teacher
2	Descriptive Statistics	Calculate basic descriptive statistics that are commonly used in data mining, and understand how they are used to summarize datasets.	Jigsaw: students will be grouped into teams and tasked to gather data from various sources, conduct descriptive statistics, and report their findings to the rest of the class.	'We learned the importance of teamwork, critical thinking, and communication skills by working together to conduct descriptive statistics on our assigned data set.'	'The goal here is to give every student a chance to delve deeper into specific aspects of data mining. As a teacher, my role is to facilitate the group and ensure everyone is participating. As students, you are expected to synthesize, analyze, and present your findings through a collaborative effort.'

ETL stands for Extract, Transform, and Load. It is a process used in data integration and data warehousing to extract data from various sources, transform it into a consistent format, and load it into a target system. The "Extract" phase involves gathering data from multiple sources, such as databases, spreadsheets, or APIs. In the "Transform" phase, the data is cleaned, validated, and standardized to ensure consistency and quality. Finally, in the "Load" phase, the transformed data is loaded into a target system, such as a data warehouse or a business intelligence tool, making it accessible for analysis and reporting. ETL is a crucial step in data management to ensure accurate and usable data for decision-making.

In the next example we are going to extract and load a data set. The data comes from here.

```
import pandas as pd
import plotly.express as px

# Extract
url = "https://raw.githubusercontent.com/businessdatasolutions/courses/main/datamining-n/d
rawDF = pd.read_csv(url).sort_values(by=['year', 'Domain'])
# Transform
rawDF = rawDF.sort_values(by=['year', 'Domain'])
rawDF.head()
```

```
                    Entity  Code  ...  Training_computation_petaflop  Domain
216                Theseus   NaN  ...                   4.000000e-14   Other
195                  SNARC   NaN  ...                            NaN   Other
198  Self Organizing System   NaN  ...                            NaN  Vision
167        Perceptron Mark I   NaN  ...                   6.950000e-10  Vision
```

```
197  Samuel Neural Checkers   NaN  ...                    4.280000e-07   Games

[5 rows x 6 columns]
```

Once you have accessed your dataset you'll want to get familiar with the content and gain insights into its quality and structure. Data analysts or data scientists collect and examine the data to understand its relevance to the project's goals. They explore the data using various techniques, such as descriptive statistics, data visualization, and data profiling. The goal is to identify patterns, relationships, and potential issues within the dataset, which helps in formulating initial hypotheses and refining the project's objectives.

```python
# Load
fig = px.scatter(rawDF, x="year", y="Training_computation_petaflop", color="Domain", log_y
fig.update_traces(marker={'size': 12})
```

```python
rawDF.describe()
```

```
       Code         year  Training_computation_petaflop
count   0.0   246.000000                   1.040000e+02
mean    NaN  2009.939024                   3.002598e+08
std     NaN    15.437310                   2.084483e+09
min     NaN  1950.000000                   4.000000e-14
25%     NaN  2010.000000                   3.650000e+00
50%     NaN  2015.500000                   2.405000e+04
75%     NaN  2019.000000                   1.100000e+07
max     NaN  2023.000000                   2.100000e+10
```

3 Lazy Learning with k-Nearest Neighbors

K-nearest neighbors is an algorithm that is commonly used in data mining. It works by identifying the k-nearest data points to a given point, and using their values to predict the value of the point in question.

Table 3.1: Lesson outline

	Topic	Tasks	Activities	Student	Teacher
2	K-Nearest Neighbors	Build a k-nearest neighbors model and explain how it may be used to predict the values of data points.	Follow along: students will participate in a guided demo of a data mining process building a model using K-Nearest Neighbors and evaluating its accuracy using a Confusion Matrix.	'We learned about the KNN algorithm, its advantages and limitations, as well as how to interpret a confusion matrix to evaluate the accuracy of a model.'	'Our goal here is to understand how data mining algorithms work and how they can be applied to real-world problems. As a teacher, my role is to clarify any doubts and ensure that everyone is actively participating. As students, you will be challenged to apply your knowledge to a problem and think critically.'

3.1 Business Case: Diagnosing Breast Cancer

Breast cancer is the top cancer in women both in the developed and the developing world. In the Netherlands it is the most pervasive form of cancer ("WHO | Cancer Country Profiles 2020" n.d.). In order to improve breast cancer outcome and survival early detection remains the most important instrument for breast cancer control. If machine learning could automate the identification of cancer, it would improve efficiency of the detection process and might also increase its effectiveness by providing greater detection accuracy.

3.2 Data Understanding

The data we will be using comes from the University of Wisconsin and is available online as an open source dataset ("UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set" n.d.). It includes measurements from digitized images from from fine-needle aspirates of breast mass. The values represent cell nuclei features.

For convenience the data in csv format is stored on Github. We can access it directly using a function for reading csv from the `pandas` library

```
url = "https://raw.githubusercontent.com/businessdatasolutions/courses/main/data%20mining/
rawDF = pd.read_csv(url)
```

Using the `info()` function we can have some basic information about the dataset.

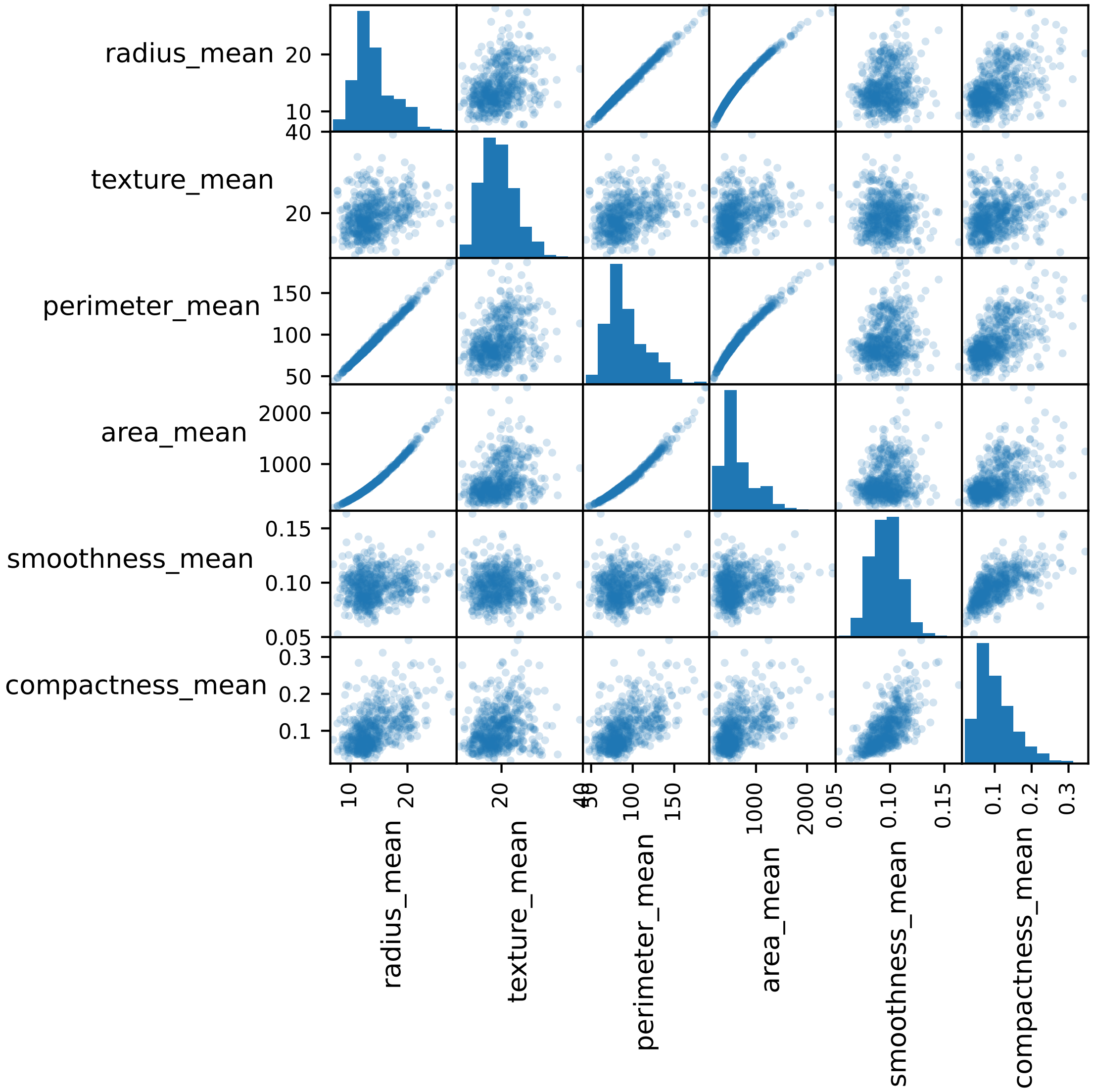
```
rawDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 569 non-null    int64
 1   diagnosis          569 non-null    object
 2   radius_mean        569 non-null    float64
 3   texture_mean       569 non-null    float64
 4   perimeter_mean     569 non-null    float64
 5   area_mean          569 non-null    float64
 6   smoothness_mean    569 non-null    float64
 7   compactness_mean   569 non-null    float64
 8   concavity_mean     569 non-null    float64
 9   points_mean        569 non-null    float64
 10  symmetry_mean      569 non-null    float64
 11  dimension_mean     569 non-null    float64
 12  radius_se          569 non-null    float64
 13  texture_se         569 non-null    float64
 14  perimeter_se       569 non-null    float64
 15  area_se            569 non-null    float64
 16  smoothness_se      569 non-null    float64
 17  compactness_se     569 non-null    float64
 18  concavity_se       569 non-null    float64
 19  points_se          569 non-null    float64
```

```
 20  symmetry_se        569 non-null    float64
 21  dimension_se       569 non-null    float64
 22  radius_worst       569 non-null    float64
 23  texture_worst      569 non-null    float64
 24  perimeter_worst    569 non-null    float64
 25  area_worst         569 non-null    float64
 26  smoothness_worst   569 non-null    float64
 27  compactness_worst  569 non-null    float64
 28  concavity_worst    569 non-null    float64
 29  points_worst       569 non-null    float64
 30  symmetry_worst     569 non-null    float64
 31  dimension_worst    569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

We can also visually look for patterns in the data.

```python
selDF = rawDF.filter(regex="mean").iloc[:, :6]
fig = scatter_matrix(selDF, alpha=0.2, figsize=(6, 6), diagonal="hist")
for ax in fig.flatten():
    ax.xaxis.label.set_rotation(90)
    ax.yaxis.label.set_rotation(0)
    ax.yaxis.label.set_ha("right")
plt.tight_layout()
plt.gcf().subplots_adjust(wspace=0, hspace=0)
plt.show()
```



Question:

- *What patterns do you observe and to what extend are they surprising?*

3.3 Preparation

The first variable, `id`, contains unique patient IDs. The IDs do not possess any relevant information for making predictions, so we will delete it from the dataset.

```python
cleanDF = rawDF.drop(["id"], axis=1)
cleanDF.head()
```

```
  diagnosis  radius_mean  ...  symmetry_worst  dimension_worst
0         B        12.32  ...          0.2827          0.06771
1         B        10.60  ...          0.2940          0.07587
2         B        11.04  ...          0.2998          0.07881
3         B        11.28  ...          0.2102          0.06784
4         B        15.19  ...          0.2487          0.06766

[5 rows x 31 columns]
```

The variable named `diagnosis` contains the outcomes we would like to predict - 'B' for 'Benign' and 'M' for 'Malignant'. The variable we would like to predict is called the 'label'. We can look at the counts for both outcomes, using the `value_counts()` function. When we set the normalize setting to `True` we get the the proportions.

```python
cntDiag = cleanDF["diagnosis"].value_counts()
propDiag = cleanDF["diagnosis"].value_counts(normalize=True)
cntDiag
```

```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

```python
propDiag
```

```
diagnosis
B    0.627417
M    0.372583
Name: proportion, dtype: float64
```

Looking again at the results from the `info()` function you'll notice that the variable `diagnosis` is coded as text (`object`). Many models require that the label is of type `category`. The `pandas` library has a function that can transform a `object` type to `category`.

```python
catType = CategoricalDtype(categories=["B", "M"], ordered=False)
cleanDF["diagnosis"] = cleanDF["diagnosis"].astype(catType)
cleanDF["diagnosis"]
```

```
0      B
1      B
2      B
3      B
4      B
      ..
564    B
565    B
566    M
567    B
568    M
Name: diagnosis, Length: 569, dtype: category
Categories (2, object): ['B', 'M']
```

The features consist of three different measurements of ten characteristics. We will take three characteristics and have a closer look.

```python
cleanDF[["radius_mean", "area_mean", "smoothness_mean"]].describe()
```

```
       radius_mean    area_mean  smoothness_mean
count   569.000000   569.000000       569.000000
mean     14.127292   654.889104         0.096360
std       3.524049   351.914129         0.014064
min       6.981000   143.500000         0.052630
25%      11.700000   420.300000         0.086370
50%      13.370000   551.100000         0.095870
75%      15.780000   782.700000         0.105300
max      28.110000  2501.000000         0.163400
```

You'll notice that the three variables have very different ranges and as a consequence `area_mean` will have a larger impact on the distance calculation than the `smootness_mean`. This could potentially cause problems for modeling. To solve this we'll apply normalization to rescale all features to a standard range of values.

We will write our own normalization function,

```python
def normalize(x):
    return (x - min(x)) / (
        max(x) - min(x)
    )  # distance of item value - minimum vector value divided by the range of all vector


testSet1 = np.arange(1, 6)
testSet2 = np.arange(1, 6) * 10


print(f"testSet1: {testSet1}\n")
```

```
testSet1: [1 2 3 4 5]
```

```python
print(f"testSet2: {testSet2}\n")
```

```
testSet2: [10 20 30 40 50]
```

```python
print(f"Normalized testSet1: {normalize(testSet1)}\n")
```

```
Normalized testSet1: [0.   0.25 0.5  0.75 1.  ]
```

```python
print(f"Normalized testSet2: {normalize(testSet2)}\n")
```

```
Normalized testSet2: [0.   0.25 0.5  0.75 1.  ]
```

and apply it to all the numerical variables in the dataframe.

```python
excluded = ["diagnosis"]  # list of columns to exclude
# X = cleanDF.loc[:, ~cleanDF.columns.isin(excluded)]
X = cleanDF.drop(excluded, axis=1)
X = X.apply(normalize, axis=0)
X[["radius_mean", "area_mean", "smoothness_mean"]].describe()
```

```
       radius_mean   area_mean  smoothness_mean
count   569.000000  569.000000       569.000000
mean      0.338222    0.216920         0.394785
std       0.166787    0.149274         0.126967
min       0.000000    0.000000         0.000000
25%       0.223342    0.117413         0.304595
50%       0.302381    0.172895         0.390358
75%       0.416442    0.271135         0.475490
max       1.000000    1.000000         1.000000
```

When we take the variables we've selected earlier and look at the summary parameters again, we'll see that the normalization was successful.

We can now split our data into training and test sets.

```
y = cleanDF["diagnosis"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

Here, X_train and y_train are the features and labels of the training data, respectively, and X_test and y_test are the features and labels of the test data.

Now we can train and evaluate our kNN model.

3.4 Modeling and Evaluation

KNN is a instance-based learning algorithm. It stores all of the training data and makes predictions based on the similarity between the input instance and the stored instances. The prediction is based on the majority class among the K nearest neighbors of the input instance.

The distance between instances is typically measured using the Euclidean distance. However, other distance measures such as the Manhattan distance or the Minkowski distance can also be used.

The pseudocode for the KNN algorithm is as follows:

To train the knn model we only need one single function from the `sklearn` library. The `fit()` function trains the model on the training data. The trained model is applied to the set with test features and the `predict()` function gives back a set of predicted values for y.

NOTE: Somehow `predict()` generates error with number columns larger than 15. This is a new issue. It used to work.

```python
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train.iloc[:,:15], y_train)
# # make predictions on the test set
```

```
KNeighborsClassifier()
```

```python
y_pred = knn.predict((X_test.iloc[:,:15]))
```

Now that we have a set of predicted labels we can compare these with the actual labels. A diffusion table shows how well the model performed.

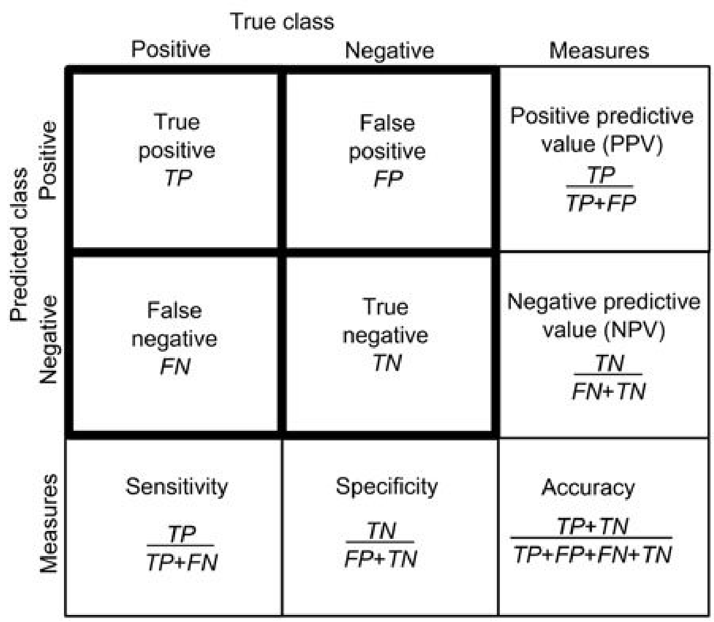


Figure 3.1: Standard diffusion table. Taken from: https://emj.bmj.com/content/emermed/36/7/431/F1.large.jp

Here is our own table:

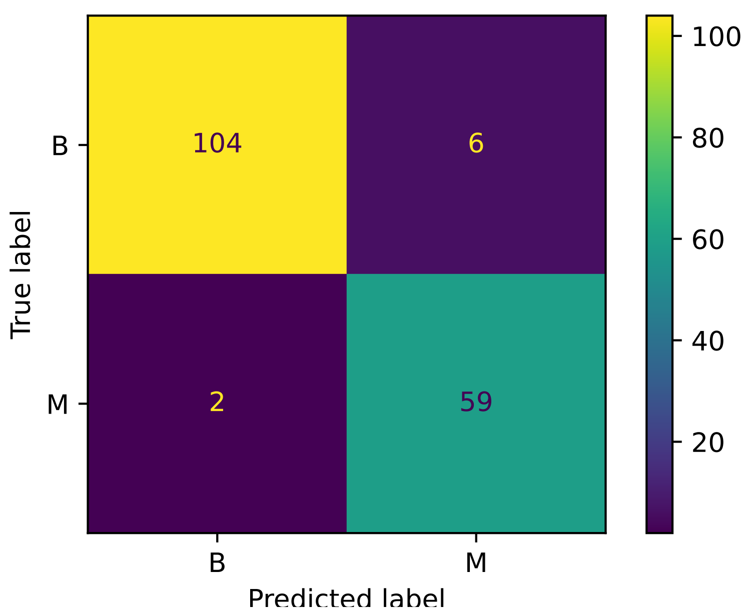
```python
cm = confusion_matrix(y_test, y_pred, labels=knn.classes_)
cm
```

```
array([[104,   6],
       [  2,  59]])
```

```python
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knn.classes_)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x140b1cd60>
```

```python
plt.show()
```



Questions:

1. *How would you assess the overall performance of the model?*
2. *What would you consider as more costly: high false negatives or high false positives levels? Why?*
3. *Try to improve the model by changing some parameters of the* `KNeighborsClassifier()` *function*

4 Probabilistic Learning with Naive Bayes Classification

Naïve Bayes is commonly used algorithm in data mining. It works by using statistical probabilities to classify data points based on their observed characteristics.

Table 4.1: Lesson outline

	Topic	Tasks	Activities	Student	Teacher
2	Naive Bayes	Build a k-nearest neighbors model and explain how it may be used to predict the values of data points.	Follow along: students will participate in a guided demo of a data mining process building a model using Naive Bayes and evaluating its accuracy using a Confusion Matrix.	'We learned about the Naive Bayes algorithm, its advantages and limitations, as well as how to interpret a confusion matrix to evaluate the accuracy of a model.'	'Our goal here is to understand how data mining algorithms work and how they can be applied to real-world problems. As a teacher, my role is to clarify any doubts and ensure that everyone is actively participating. As students, you will be challenged to apply your knowledge to a problem and think critically.'

4.1 Business Case: Filtering Spam

In 2020 spam accounted for more than 50% of total e-mail traffic ("Spam Statistics: Spam e-Mail Traffic Share 2019" n.d.). This illustrates the value of a good spam filter. Naive Bayes spam filtering is a standard technique for handling spam. It is one of the oldest ways of doing spam filtering, with roots in the 1990s.

4.2 Data Understanding

The data you'll be using comes from the SMS Spam Collection ("UCI Machine Learning Repository: SMS Spam Collection Data Set" n.d.). It contains a set of SMS messages that are labeled 'ham' or 'spam'. and is a standard data set for testing spam filtering methods.

```
url = "https://raw.githubusercontent.com/businessdatasolutions/courses/main/datamining-n/d
rawDF = pd.read_csv(url)
rawDF.head()
```

```
   type                                               text
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
```

The variable `type` is of class `object` which in Python refers to text. As this variable indicates whether the message belongs to the category ham or spam it is better to convert it to a `category` variable.

```
catType = CategoricalDtype(categories=["ham", "spam"], ordered=False)
rawDF.type = rawDF.type.astype(catType)
rawDF.type
```

```
0        ham
1        ham
2       spam
3        ham
4        ham
        ...
5567    spam
5568     ham
5569     ham
5570     ham
5571     ham
Name: type, Length: 5572, dtype: category
Categories (2, object): ['ham', 'spam']
```

To see how the types of sms messages are distributed you can compare the counts for each category.

```
rawDF.type.value_counts()
```

```
type
ham     4825
spam     747
Name: count, dtype: int64
```

Often you'll prefer the relative counts.

```
rawDF.type.value_counts(normalize=True)
```

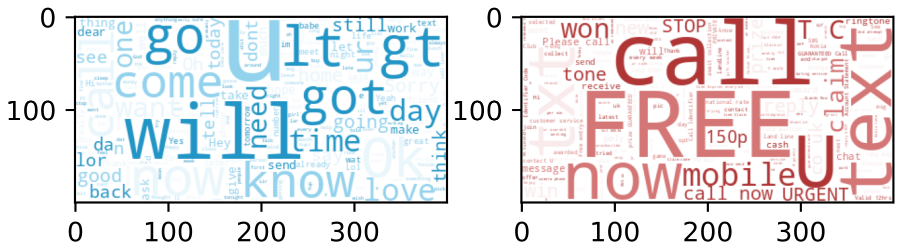
```
type
ham     0.865937
spam    0.134063
Name: proportion, dtype: float64
```

You can also visually inspect the data by creating wordclouds for each sms type.

```
# Generate a word cloud image]
hamText = ' '.join([Text for Text in rawDF[rawDF['type']=='ham']['text']])
spamText = ' '.join([Text for Text in rawDF[rawDF['type']=='spam']['text']])
colorListHam=['#e9f6fb','#92d2ed','#2195c5']
colorListSpam=['#f9ebeb','#d57676','#b03636']
colormapHam=colors.ListedColormap(colorListHam)
colormapSpam=colors.ListedColormap(colorListSpam)
wordcloudHam = WordCloud(background_color='white', colormap=colormapHam).generate(hamText)
wordcloudSpam = WordCloud(background_color='white', colormap=colormapSpam).generate(spamTe

# Display the generated image:
# the matplotlib way:
fig, (wc1, wc2) = plt.subplots(1, 2)
fig.suptitle('Wordclouds for ham and spam')
wc1.imshow(wordcloudHam)
wc2.imshow(wordcloudSpam)
plt.show()
```

Wordclouds for ham and spam



Question:

- *What differences do you notice?*

4.3 Preparation

After you've glimpsed over the data and have a certain understanding of its structure and content, you are now ready to prepare the data for further processing. For the naive bayes model you'll need to have a dataframe with wordcounts. To save on computation time you can set a limit on the number of features (columns) in the wordsDF dataframe.

```
vectorizer = TfidfVectorizer(max_features=1000)
vectors = vectorizer.fit_transform(rawDF.text)
wordsDF = pd.DataFrame(vectors.toarray(), columns=vectorizer.get_feature_names_out())
wordsDF.head()
```

	000	03	04	0800	08000839402	...	your	yours	yourself	yr	yup
0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	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    0.0        0.0  0.0  0.0

[5 rows x 1000 columns]
```

The counts are normalized in such a way that the words that are most likely to have predictive power get heavier weights. For instance stopword like *"a"* and *"for"* most probably will equally likely feature in spam as in ham messages. Therefore these words will be assigned lower normalized counts.

Before we start modeling we need to split all datasets into *train* and *test* sets. The function *train_test_split()* can be used to create balanced splits of the data. In this case we'll create a 75/25% split.

```
xTrain, xTest, yTrain, yTest = train_test_split(wordsDF, rawDF.type)
```

4.4 Modeling and Evaluation

We have now everything in place to start training our model and evaluate against our test dataset. The `MultinomialNB().fit()` function is part of the `scikit learn` package. It takes in the features and labels of our training dataset and returns a trained naive bayes model.

```
bayes = MultinomialNB()
bayes.fit(xTrain, yTrain)
```

```
MultinomialNB()
```

The model can be applied to the test features using the `predict()` function which generates a array of predictions. Using a confusion matrix we can analyze the performance of our model.

```
yPred = bayes.predict(xTest)
yTrue = yTest
```

```
accuracyScore = accuracy_score(yTrue, yPred)
print(f'Accuracy: {accuracyScore}')
```

```
Accuracy: 0.9741564967695621
```

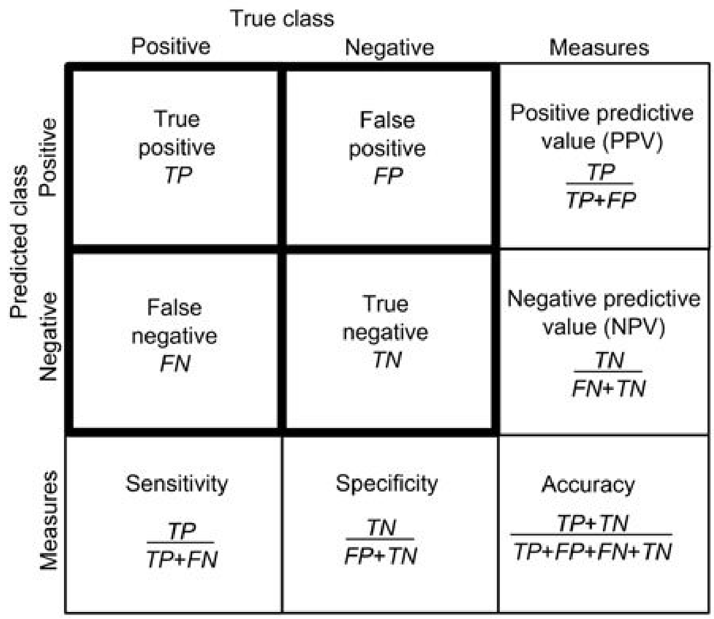


Figure 4.1: Standard diffusion table. Taken from: https://emj.bmj.com/content/emermed/36/7/431/F1.large.jp

```
matrix = confusion_matrix(yTrue, yPred)
labelNames = pd.Series(['ham', 'spam'])
pd.DataFrame(matrix,
     columns='Predicted ' + labelNames,
     index='Is ' + labelNames)
```

```
         Predicted ham  Predicted spam
Is ham            1199               3
Is spam             33             158
```

Questions:

1. *What do you think is the role of the* `alpha` *parameter in the* `MultinomialNB()` *function?*
2. *How would you assess the overall performance of the model?*
3. *What would you consider as more costly: high false negatives or high false positives levels? Why?*

References

"Spam Statistics: Spam e-Mail Traffic Share 2019." n.d. *Statista*. Accessed January 10, 2021. https://www.statista.com/statistics/420391/spam-email-traffic-share/.

"UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set." n.d. Accessed January 7, 2021. https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic).

"UCI Machine Learning Repository: SMS Spam Collection Data Set." n.d. Accessed January 9, 2021. https://archive.ics.uci.edu/ml/datasets/sms+spam+collection.

"WHO | Cancer Country Profiles 2020." n.d. *WHO*. Accessed January 7, 2021. http://www.who.int/cancer/country-profiles/en/.