Recap

- **Classification** aims at predicting a **nominal target feature** based on **one or multiple** other **(numerical) features**

- **Logistic regression** as a simple-yet-popular classification method that builds on multiple linear regression

- **One-vs-rest classification** enables a binary classification method to classify data points into more than two classes

- **Precision, recall, and F1** as prediction quality measures that can be **micro- or macro-averaged** when dealing with more than two classes
3.3 k-Nearest-Neighbors

- k-Nearest-Neighbors (kNN) is another simple-yet-popular classification method that often serves as a baseline

- kNN is a so-called lazy learning method, which reflects that it does not actually learn the parameters of a model, but always looks at the training data
  - no cost for training a model, i.e., learning parameters
  - cost at runtime (i.e., when classifying a data point) depends on the amount of training data available
k-Nearest-Neighbors

- To classify a previously unseen data point, kNN.
  - Identifies the \( k \) closest data points in the training data according to a suitable distance measure.
  - Predicts the nominal target feature (class) as the value that is most frequent among the \( k \) closest data points.
k-Nearest-Neighbors

$\text{Prediction: } \times$

$k = 3$
Minkowski Distance

- **Minkowski Distance** as a suitable distance measure

\[ d(x, x') = \left( \sum_{i=1}^{m} |x_i - x'_i|^p \right)^{1/p} \]

with \( p \) as a constant

- Minkowski Distance is a **metric**, i.e., it is
  - **non-negative** \( \forall x, x' : d(x, x') \geq 0 \)
  - **symmetric** \( \forall x, x' : d(x, x') = d(x', x) \)
  - **subadditive** \( \forall x, x', x'' : d(x, x'') \leq d(x, x') + d(x', x'') \)
    (triangle inequality)
**Manhattan Distance** as Minkowski Distance with $p = 1$

$$d(x, x') = \sum_{i=1}^{m} |x_i - x'_i|$$

Distance: 5 Blocks (for a taxicab)
Euclidean Distance

- **Euclidean Distance** as Minkowski Distance with $p = 2$

$$d(x, x') = \sqrt{\sum_{i=1}^{m} (x_i - x'_i)^2}$$

Distance: 4.12 Blocks (for a bird)
Normalization and Standardization

- When computing Minkowski Distances, the magnitudes of features matter, e.g.:
  - Car 1 with 100 hp weighting 2000 lbs
  - Car 2 with 100 hp weighting 2200 lbs
  - Car 3 with 300 hp weighting 2000 lbs

  Car 1 has the same distance from Car 2 as from Car 3

- To avoid that features with larger magnitude (i.e., generally larger values) dominate the distances computed, it makes sense to normalize or standardize features upfron
Min-Max Normalization maps feature values onto $[0, 1]$.

- Let $z_1, \ldots, z_n$ be the feature values observed in the data.
- The transformed feature value is then obtained as

$$z'_i = \frac{z_i - \min_j(z_j)}{\max_j(z_j) - \min_j(z_j)}$$

- Minimum is mapped to 0; maximum is mapped to 1.

- Min-Max Normalization is sensitive to outliers in the data.
Standardization transforms feature values, so that they reflect by how many standard deviations a value deviates from the mean observed in the data.

Let \( z_1, \ldots, z_n \) be the feature values observed in the data.

The transformed feature value is then obtained as

\[
\tilde{z}_i = \frac{z_i - \mu}{\sigma}
\]

with

\[
\mu = \frac{1}{n} \sum_{i=1}^{n} z_i \quad \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (z_i - \mu)^2}
\]
Predicting Origin from Power and Weight

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv("/path/to/auto-mpg.data", sep="\s+", header=None)

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin (0:Non-U.S. / 1:U.S.) as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# use kNN with k = 3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_predicted = knn.predict(X_test)

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))  # 0.658227848101
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv("/path/to/auto-mpg.data", sep="\s+", header=None)

...  

# normalize data
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X_train)  # determine min and max
X_train_normalized = min_max_scaler.transform(X_train)
X_test_normalized = min_max_scaler.transform(X_test)

# use kNN with k = 3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_normalized, y_train)
y_predicted = knn.predict(X_test_normalized)

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))  # 0.696202531646
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv("/path/to/auto-mpg.data", sep="\s+", header=None)
...

# normalize data
scaler = StandardScaler()
scaler.fit(X_train)  # determine mean and standard deviation
X_train_normalized = scaler.transform(X_train)
X_test_normalized = scaler.transform(X_test)

# use kNN with k = 3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_normalized, y_train)
y_predicted = knn.predict(X_test_normalized)

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))  # 0.683544303797
3.4 Naïve Bayes

- **Naïve Bayes** is a classification method that is often used for **text classification** (e.g., e-mails as SPAM or HAM), but variants for arbitrary data exist.

- Naïve Bayes supports an **arbitrary number of classes**.

- The name “Naïve Bayes” refers to the fact that **Bayes’ theorem** is used and that a (naïve) **independence assumption** is made about the data.
Let’s consider two events $A$ and $B$

- $A$ is the event that an object is a circle
- $B$ is the event that an object is green

$P[A] = \frac{5}{9}$  \hspace{1cm}  $P[B] = \frac{4}{9}$

We refer to $A \land B$ as the joint event that an object is a green circle

$P[A \land B] = P[A, B] = \frac{3}{9}$
The **conditional probability** $P[B|A]$ ($B$ given $A$) is the probability that the event $B$ occurs if we already know that the event $A$ has occurred.

\[
P[B|A] = \frac{P[A \land B]}{P[A]}
\]

Here:

\[
P[B|A] = \frac{3}{5}
\]

\[
P[A|B] = \frac{3}{4}
\]
Independence

- Two events $A$ and $B$ are called (stochastically) **independent**, if the following holds for their joint probability

$$P[A \land B] = P[A] \cdot P[B]$$

- In our example, the events $A$ and $B$ are **not independent**

$$\frac{3}{9} \neq \frac{5}{9} \cdot \frac{4}{9}$$
Bayes’ Theorem

- **Thomas Bayes** (1701-1761) famously observed the following *theorem* regarding the *conditional probabilities* of events

\[
P[A | B] = \frac{P[B | A] P[A]}{P[B]}\]

- **Bayes’ theorem** is particularly useful when, for two events \(A\) and \(B\), *one* of the conditional probabilities is *easy* to estimate, but *the other* is *hard* to estimate
Bayes’ Theorem in Action

- **Example**: Examining animals in the wild
  - $A$ is the event that the animal is a fox
  - $B$ is the event that the animal has rabies (“Tollwut”)

- Assume that we know the following probabilities
  - $P[A] = 0.1$ (e.g., estimated based on video surveillance)
  - $P[B] = 0.05$ (e.g., estimated based on hunted animals)
  - $P[A|B] = 0.25$ (e.g., estimated based on deceased animals)

- We can now estimate the probability that a fox has rabies

\[
P[B | A] = \frac{0.25 \cdot 0.05}{0.1} = 0.125
\]
Bag-of-Words Model

- Common **preprocessing steps** for text documents
  - convert all letters to **lower case**
  - remove **stop words** (e.g., a, the, or, of)
  - split documents at **white spaces** (e.g., _, \n, \t) and **punctuation marks** (e.g., ? ! : , ;)

- The document is then viewed as a **bag of words** (i.e., a multiset preserving frequency information)

```
The green politician Peter Green
```

```
{ green, green, peter, politician }
```
Data Matrix

- Documents as bags-of-words with their respective class can be viewed as a **data matrix**

- **Example**: Five documents $d_1, \ldots, d_5$ consisting of words $a, b, x, y$ and belonging to either class **Spam** or **Ham**

<table>
<thead>
<tr>
<th></th>
<th>Words</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$d_1$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$d_2$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d_3$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$d_4$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$d_5$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Naïve Bayes for Text Classification

- For a previously unseen document $d$ and any class $c$, we need to estimate the conditional probability

$$P[c \mid d]$$

that $c$ is the correct class for document $d$

- The document is then assigned to the class $c$, which has the highest probability
Naïve Bayes for Text Classification

- Bayes’ theorem allows rewriting the conditional probability as

\[
P[c \mid d] = \frac{P[d \mid c] \cdot P[c]}{P[d]} \]

since \( P[d] \) is constant for any document

\[
P[c \mid d] \propto P[d \mid c] \cdot P[c] \]

with

- \( P[c] \) as the so-called class prior
- \( P[d \mid c] \) as the probability that document \( d \) is from class \( c \)

- How can we estimate these probabilities?
Class Priors in Naïve Bayes

- **Class priors** can be estimated based on the training data as

\[
P[c] = \frac{\text{\# Documents from Class } c}{\text{\# Documents}}
\]

- **Example:**

<table>
<thead>
<tr>
<th></th>
<th>Words</th>
<th></th>
<th></th>
<th></th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>x</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>d₁</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>H</td>
</tr>
<tr>
<td>d₂</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>S</td>
</tr>
<tr>
<td>d₃</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>d₄</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>S</td>
</tr>
<tr>
<td>d₅</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>H</td>
</tr>
</tbody>
</table>

\[P[H] = \frac{3}{5}\]
\[P[S] = \frac{2}{5}\]
The **conditional probability** that document $d$ is from class $c$ is estimated based on the **contained words** as

$$P[d | c] \propto \prod_{w \in d} P[w | c] f(w,d)$$

with $f(w,d)$ as the frequency of word $w$ in document $d$ and

$$P[w | c] = \frac{\# \text{ Occurrences of } w \text{ in Document from Class } c}{\# \text{ Word Occurrences in Documents from Class } c}$$

as the probability that we randomly draw the word $w$ from the documents in class $c$. 

---

**Conditional Probabilities in Naïve Bayes**
Intuitively, the conditional probability

\[
P [ d \mid c ] \propto \prod_{w \in d} P [ w \mid c ] f(w,d)
\]

corresponds to the probability that we randomly draw exactly the words in \( d \) when drawing words from documents in \( c \).

Note that we make the **simplifying assumption** that words occur **independently** from each other (hence the product), which explains the label “naïve”
Conditional Probabilities in Naïve Bayes

**Example:**

<table>
<thead>
<tr>
<th></th>
<th>Words</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a  b  x  y</td>
<td></td>
</tr>
<tr>
<td>$d_1$</td>
<td>2  1  0  0</td>
<td>H</td>
</tr>
<tr>
<td>$d_2$</td>
<td>0  1  2  2</td>
<td>S</td>
</tr>
<tr>
<td>$d_3$</td>
<td>2  1  1  1</td>
<td>H</td>
</tr>
<tr>
<td>$d_4$</td>
<td>1  0  2  2</td>
<td>S</td>
</tr>
<tr>
<td>$d_5$</td>
<td>1  1  0  0</td>
<td>H</td>
</tr>
</tbody>
</table>

\[
P[a \mid H] = \frac{5}{10} \quad P[a \mid S] = \frac{1}{10}
\]
\[
P[b \mid H] = \frac{3}{10} \quad P[b \mid S] = \frac{1}{10}
\]
\[
P[x \mid H] = \frac{1}{10} \quad P[x \mid S] = \frac{4}{10}
\]
\[
P[y \mid H] = \frac{1}{10} \quad P[y \mid S] = \frac{4}{10}
\]
Putting Naïve Bayes to Use

- Let’s classify a **previously unseen document**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>x</th>
<th>y</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_6$</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>

- We obtain the following probabilities

\[
P[H \mid d_6] = P[d_6 \mid H] \cdot P[H] = \left( \frac{5}{10} \cdot \frac{5}{10} \cdot \frac{3}{10} \cdot \frac{3}{10} \cdot \frac{1}{10} \cdot \frac{1}{10} \right) \cdot \frac{3}{5} = \frac{135}{10^6}
\]

\[
P[S \mid d_6] = P[d_6 \mid S] \cdot P[S] = \left( \frac{1}{10} \cdot \frac{1}{10} \cdot \frac{1}{10} \cdot \frac{1}{10} \cdot \frac{4}{10} \cdot \frac{4}{10} \right) \cdot \frac{2}{5} = \frac{6.4}{10^6}
\]

and thus classify the document as **Ham**
When implementing probabilistic methods like Naïve Bayes, it is often useful to apply a logarithmic transformation to avoid underflows and other numerical issues.

\[
\log P[c \mid d] \propto \log P[d \mid c] + \log P[c]
\]

\[
\log P[d \mid c] \propto \sum_{w \in d} f(w, d) \cdot \log P[w \mid c]
\]
Classifying Reviews with Naïve Bayes

- We’ll work on a dataset of more than 400,000 reviews from Amazon about unlocked mobile phones
  https://www.kaggle.com/.../amazon-...-phones

- Example:

  """CLEAR CLEAN ESN"" Sprint EPIC 4G Galaxy SPH-D700*FRONT CAMERA*ANDROID*SLIDER*QWERTY KEYBOARD*TOUCH SCREEN",Samsung,199.99,4,"nice phone, nice up grade from my pantach revue. Very clean set up and easy set up. never had an android phone but they are fantastic to say the least. perfect size for surfing and social media. great phone samsung",0

  We want to predict the rating (1-5) from the content
import math
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

# load data
reviews = pd.read_csv("/path/to/amazon-unlocked-mobile.csv", encoding='utf-8')

X = reviews.iloc[:,4].values
X_clean = X[pd.notnull(X)]
y = reviews.iloc[:,3].values
y_clean = y[pd.notnull(X)]

# convert documents into bags-of-words
vectorizer = CountVectorizer()
X_cnt = vectorizer.fit_transform(X_clean)

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X_cnt, y_clean, test_size=0.2, random_state=0)

# train naive bakes classifier
nb = MultinomialNB(alpha=0.0)
nb.fit(X_train, y_train)
Classifying Reviews with Naïve Bayes

# predict labels
y_predicted = nb.predict(X_test)

# compute confusion matrix
print(confusion_matrix(y_true=y_test, y_pred=y_predicted))

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))

# print class priors
for c in range(0, len(nb.classes_)):
    print('Class:' + str(c))
    print(str(math.exp(nb.class_log_prior_[c])))

# print probabilities per class for words in {android, apple, ... , error, crash}
feature_names = vectorizer.get_feature_names()
for w in ['android', 'apple', 'good', 'bad', 'terrible', 'error', 'crash']:
    print('Word:' + w)
    for c in range(0, len(nb.classes_)):
        print(str(c) + ' : ' + str(math.exp(nb.coef_[c][feature_names.index(w)])))
Summary

- **k-Nearest-Neighbors** as a lazy classification method that classifies a previously unseen data point based on the classes of the $k$ closest data points from the training data.

- **Naïve Bayes** as a probabilistic classification method that is primarily used for text classification, but can be extended to deal with other data.
References