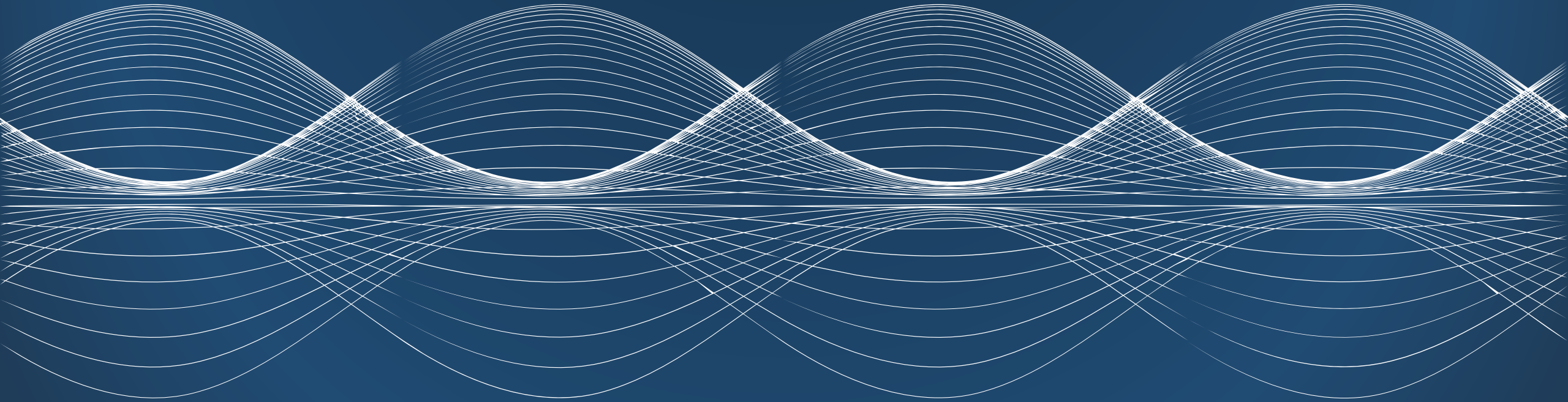


Naïve Bayes Classification

Advanced Research Topics – 7PAM2016

Dr Vanessa Graber (based on slides by Dr William Alston)

Learning outcomes

After this lecture, you will:

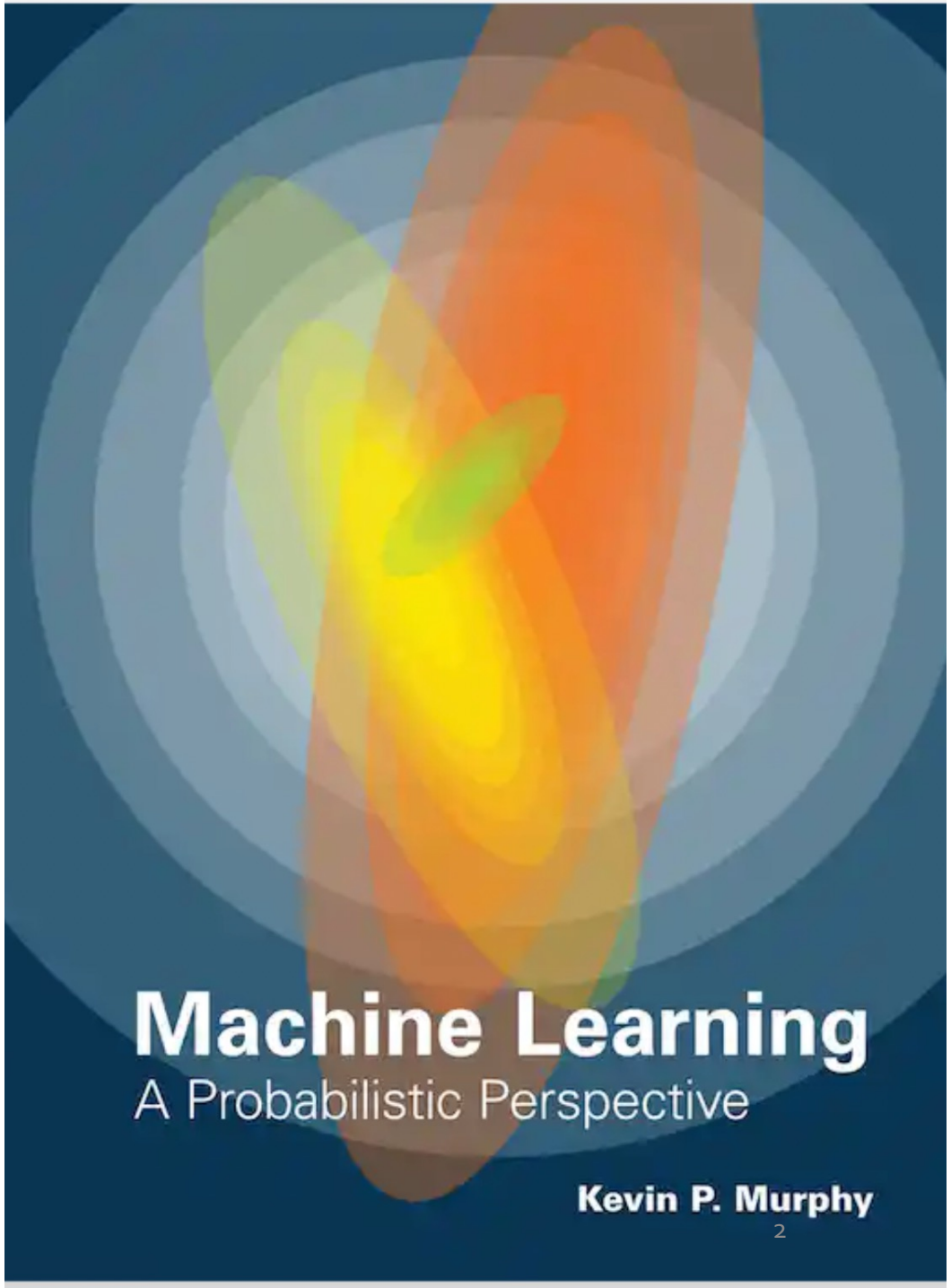
- Understand generative methods for classification.
- Be able to make **probabilistic classifications** on data containing two or more classes.
- Use the Naïve Bayes assumption for classification problems.
- Be able to implement this approach in Python.

Following this book:

Probabilistic Machine Learning – Chapter 9

Kevin P. Murphy 2022

https://herts.instructure.com/courses/103289/files/6446125?module_item_id=2844135



Introduction

Probabilistic classification

Using a Naïve Bayes classifier

Research examples

Summary

Introduction

Probabilistic classification

Using a Naïve Bayes classifier

Research examples

Summary

Supervised learning

Two types of approaches

In **classification**, the label is discrete,
while in **regression**, the label is continuous.

- For example, in astronomy, the task of determining whether an object is a star, a galaxy, or a quasar is a classification problem: the label is from three distinct categories.

- On the other hand, we might wish to estimate the age of an object based on such observations: this would be a regression problem, because the label (age) is a continuous quantity.

Example of classification: spam detection

- Input: email
- Output: spam/ham
- Setup:
  - Get a large collection of example emails, each labeled “spam” or “ham”
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future emails
- Features: The attributes used to make the ham / spam decision
  - Words: FREE!
  - Text Patterns: $dd, CAPS
  - Non-text: SenderInContacts
  - …

Dear Sir.

First, I must solicit your confidence in this transaction, this is by virture of its nature as being utterly confidencial and top secret. …

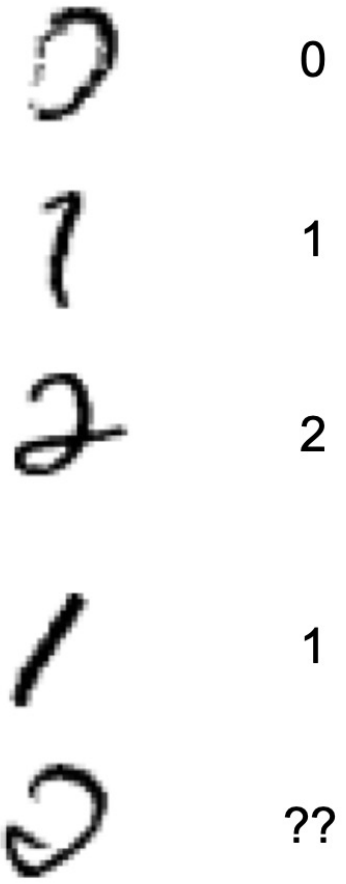
TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.

99 MILLION EMAIL ADDRESSES FOR ONLY $99

Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

Example of classification: handwritten digits

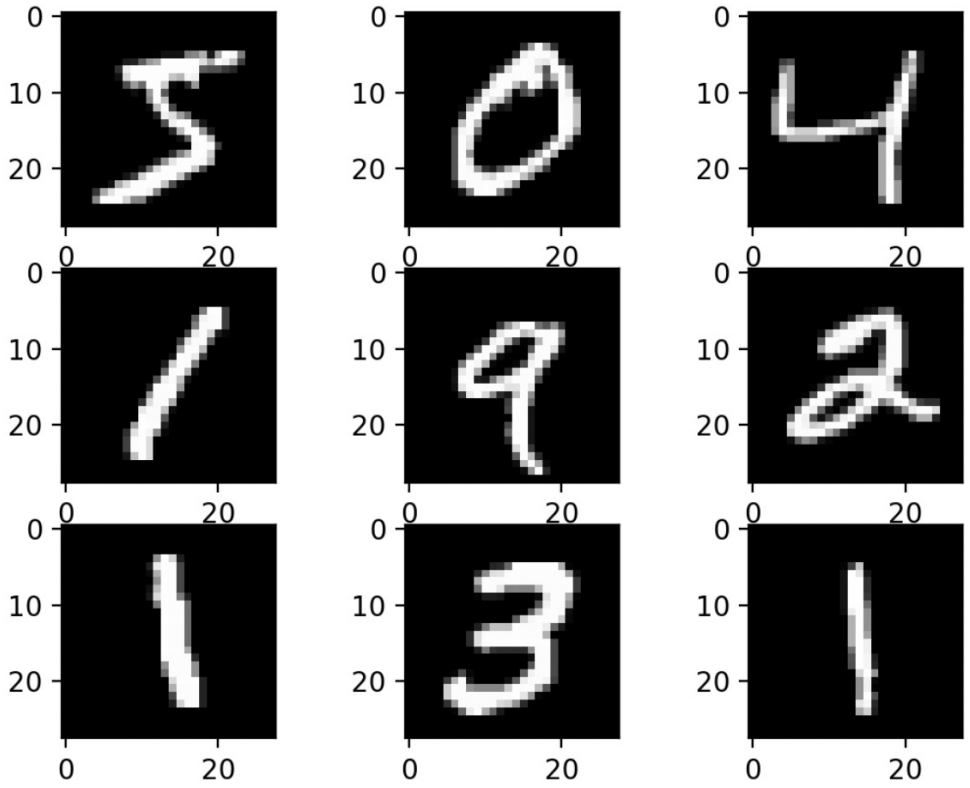
- Input: images / pixel grids
- Output: a digit 0-9
- Setup:
  - Get a large collection of example images, each labeled with a digit
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future digit images
- Features: The attributes used to make the digit decision
  - Pixels: (6,8)=ON
  - Shape Patterns: NumComponents, AspectRatio, NumLoops
  - …



Example of classifica-tion: hand-written digits

Benchmarking problem in image processing:

Handwritten digits classification with the MNIST (Modified National Institute of Standards and Technology) dataset



Other classification problems

- In classification, we predict labels, y, (our classes) for inputs, x.

- Further examples:
  - Optical character recognition OCR (input: images, classes: characters)
  - Medical diagnosis (input: symptoms, classes: diseases)
  - Automatic essay grader (input: document, classes: grades)
  - Fraud detection (input: account activity, classes: fraud / no fraud)
  - Customer profile (input: browsing activity, classes: consumer type)
  - … many more

Classification is an **important commercial technology**!

Classification concepts

- **Data:** labelled instances, e.g., emails marked spam/ham
  - Subsets: training set, validation set, test set
- **Features:** attribute-value pairs which characterise each x
- **Experimentation cycle:**
  - Learn parameters (e.g., model probabilities) on training set
  - Tune hyperparameters on validation set
  - Compute accuracy on the test set at the very end
  - Very important: never "peek" at the test set!
- **Evaluation:** accuracy (fraction of instances predicted correctly)
- **Overfitting and generalisation:**
  - Classifier should perform well on the (unseen) test data
  - Overfitting: fit training data very closely but not generalising well

Introduction

Probabilistic classification

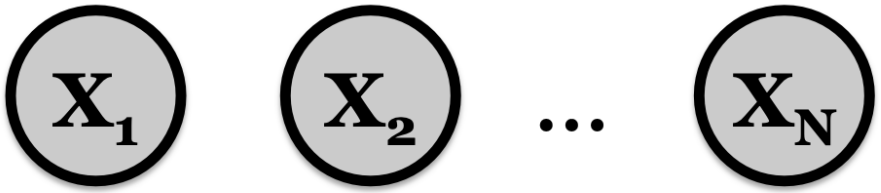
Using a Naïve Bayes classifier

Research examples

Summary

Key idea

- Each input **x** is defined by a set of features x_i.
- Each feature x_i is one (typically observed) **random variable**:



- The class label y is a **hidden random variable**:

- **Task**: return the most likely class y* for the input item **x**:

$$y^* = \text{argmax}_y P(y \mid \mathbf{x}) = \text{argmax}_y P(y \mid x_1 \ldots x_N)$$

What does this mean?

- For probabilistic classification, test **predictions take the form of class probabilities**. This contrasts with methods which provide only a guess at the class label. This distinction is analogous to the difference between predictive distributions and point predictions in regression.

> Since generalisation to test cases inherently involves some level of uncertainty, it seems natural to make predictions in a way that reflects these uncertainties.

- In a practical application, we may seek a class guess. This guess can be obtained as the solution to a **decision problem**, involving the predictive probabilities as well as a specification of the consequences of making specific predictions (encoded in the loss function).

Discriminative vs generative

- The natural starting point for classification is the **joint probability** P(y, x), where y denotes the class label. Using Bayes' theorem, this joint probability can be decomposed in the following ways:

  - P(x) P(y|x) - **discriminative**:
    e.g., Gaussian Process classification (see lecture 6)
  - P(y) P(x|y) - **generative**: this lecture

- The generative approach models the class-conditional distributions P(x|y) for $y = y_1, \ldots, y_M$ and the class prior probabilities P(y), and then computes the posterior probability P(y|x) for each class using the input prior probabilities P(x).

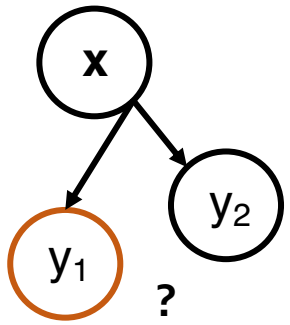
Returning to the key idea

- The features x_i are observed **random variables,** while y is the hidden random variable, we want to determine.

- **Task**: return the most likely class y* for the input item **x**:

$$y^* = \text{argmax}_y P(y \mid \mathbf{x}) = \text{argmax}_y P(y \mid x_1 \ldots x_N)$$

To perform classification, we combine a probabilistic model (you already know how to do this kind of inference) with a decision rule (e.g., pick the most probable hypothesis) to compute most likely y.



Example: binary classification

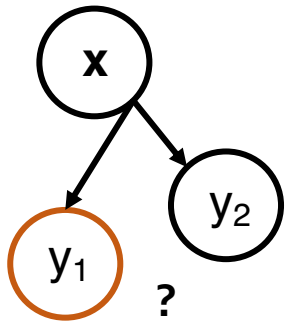
- We want to determine if an input x, belongs to a class y_1 or y_2. We first have to **determine the two conditional probabilities**:

$$P(y_j|x_1 \dots x_N) = \frac{P(x_1 \dots x_N|y_j)P(y_j)}{P(x_1 \dots x_N)}$$

- Unless we have extraordinarily large amounts of examples, we typically cannot determine $P(x_1 \dots x_N|y_j)$. The **problem is intractable**.

- However, the above equation assumes (the most general case) that for a given label y_j the **features x_i depend on each other**.

Naïve Bayes model



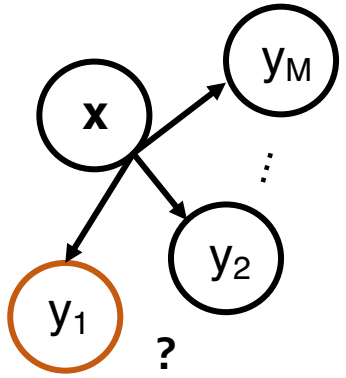
- We can **significantly simplify** by assuming that **features are independent given a label.** Then:

$$P(y_j|x_1 \dots x_N) = \frac{P(x_1 \dots x_N|y_j)P(y_j)}{P(x_1 \dots x_N)}$$

$$= \frac{P(x_1|y_j) \dots P(x_N|y_j)P(y_j)}{P(x_1 \dots x_N)}$$

$$= \frac{\prod_{i=1}^{N} P(x_i|y_j)\,P(y_j)}{P(x_1 \dots x_N)} \propto \prod_{i=1}^{N} P(x_i|y_j)P(y_j)$$

- The term in the denominator only serves to correctly normalise the probabilities. As the term does not depend on our class labels, we can drop it for classification tasks.

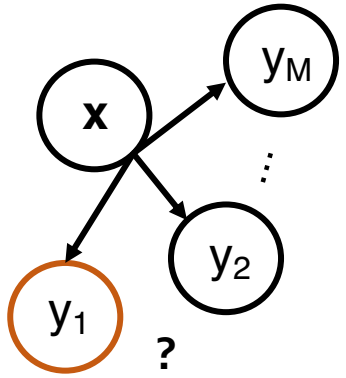


$$P(y_j|x_1 \dots x_N) \propto \prod_{i=1}^{N} P(x_i|y_j)P(y_j)$$
$$= P(y_j, x_1 \dots x_N)$$

General naïve Bayes model

- The right rand side is equivalent to the **joint probability** P(y, x) that we saw earlier.
- The key advantage: **we only have to specify how each feature depends on the class**.
- This reduces the complexity of the problem significantly from $|y|*|x|^N$ to $N*|x|*|y|*|y|$.

For a naïve Bayes' model, the total number of parameters is linear in N.



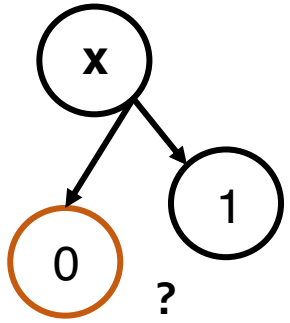
- To turn our probabilistic model into a classifier, we have to add a decision rule and then return the class label y*:

$$y^* = \text{argmax}_y \, P(y_j | x_1 \dots x_N)$$
$$= \text{argmax}_y \prod_{i=1}^{N} P(x_i | y_j) P(y_j)$$

General naïve Bayes classifier

To classify our data, we need to estimate the categorical distribution $P(y_j)$ as well as the conditional probability $P(x_i | y_j)$ for each(!) feature x_i and each(!) class y_j.

Naïve Bayes: two classes

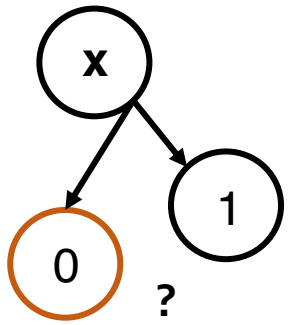


$$y^* = \text{argmax}_y \prod_{i=1}^{N} P(x_i|y_j)P(y_j)$$

- Let us assume, that we have two classes $y \in \{0,1\}$. Then, we predict that x belongs to the class y=1, iff (if and only if):

$$\frac{\prod_{i=1}^{N} P(x_i|y=1)P(y=1)}{\prod_{i=1}^{N} P(x_i|y=0)P(y=0)} > 1$$

Naïve Bayes: two classes



- Let us also assume that our input **x** is charac- terised by **Boolean features**, i.e., $x_i \in \{0,1\}$. In this case, we can **define the probabilities**:

$$p_i \equiv P(x_i = 1|y = 1)$$
$$(1 - p_i) = P(x_i = 0|y = 1)$$

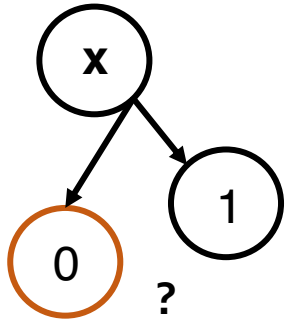
$$q_i \equiv P(x_i = 1|y = 0)$$
$$(1 - q_i) = P(x_i = 0|y = 0)$$

- We can write this in a **more compact form** (using the Bernoulli distribution):

$$P(x_i|y = 1) = {p_i}^{x_i}(1 - p_i)^{1-x_i}$$

$$P(x_i|y = 0) = {q_i}^{x_i}(1 - q_i)^{1-x_i}$$

Naïve Bayes: two classes

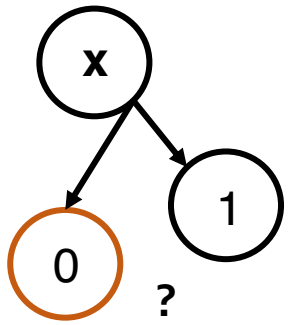


- Combining results from previous two slides:

$$1 < \frac{\prod_{i=1}^{N} P(x_i|y=1)P(y=1)}{\prod_{i=1}^{N} P(x_i|y=0)P(y=0)}$$

$$= \frac{\prod_{i=1}^{N} {p_i}^{x_i}(1-p_i)^{1-x_i}\, P(y=1)}{\prod_{i=1}^{N} {q_i}^{x_i}(1-q_i)^{1-x_i}\, P(y=0)}$$

$$= \frac{P(y=1)}{P(y=0)} \frac{\prod_{i=1}^{N}(1-p_i)\left(\frac{p_i}{1-p_i}\right)^{x_i}}{\prod_{i=1}^{N}(1-q_i)\left(\frac{q_i}{1-q_i}\right)^{x_i}}$$



Naïve Bayes: a linear classifier

- **Taking the logarithm** of the last expression, we can predict that y=1 iff:

$$\log\frac{P(y=1)}{P(y=0)}+\sum_{i=1}^{N}\log\frac{1-p_i}{1-q_i}+\sum_{i=1}^{N}x_i\left(\log\frac{p_i}{1-p_i}-\log\frac{q_i}{1-q_i}\right)>0$$

- Naïve Bayes is a **linear classifier**!
- The **feature weight**, w_i, is defined as

$$w_i \equiv \log\frac{p_i}{1-p_i}-\log\frac{q_i}{1-q_i}=\log\frac{p_i}{q_i}\frac{1-q_i}{1-p_i}$$

If $p_i = q_i$ then w_i=0 and the feature is irrelevant.

Learning = parameter estimation

- Classifiers based on probabilistic models such as naïve Bayes classifiers are defined by probability distributions.

In these probabilistic approaches, learning usually means estimating the parameters of the model's distributions. This is typically straight forward.

- Note however that learning the underlying structure of a probabilistic model (i.e., how different variables depend on each other) would be a much harder task.

Supervised learning

$$y^* = \text{argmax}_y \prod_{i=1}^{N} P(x_i|y_j)P(y_j)$$

- If we have **D labelled training items** in our training dataset, then the **class probability** can be computed as

$$P(y = y_j) = \text{freq}(y = y_j)/D$$

where $\text{freq}(y = y_j)$ denotes number of items with class label y_j.

- Similarly, our class-conditional probabilities can be obtained as

$$P(x_i = \tilde{x}|y = y_j) = \text{freq}(x_i = \tilde{x}|y = y_j)/\text{freq}(y = y_j)$$

Here, $\text{freq}(x_i = \tilde{x}|y = y_j)$ is number of items with label y_j & feature $x_i = \tilde{x}$.

Example: will we play tennis today?

- Let's say we have a table with information on the conditions (outlook of weather, temperature, humidity, wind strength) under which we play tennis:

	O	T	H	W	Play?
1	S	H	H	W	-
2	S	H	H	S	-
3	O	H	H	W	+
4	R	M	H	W	+
5	R	C	N	W	+
6	R	C	N	S	-
7	O	C	N	S	+
8	S	M	H	W	-
9	S	C	N	W	+
10	R	M	N	W	+
11	S	M	N	S	+
12	O	M	H	S	+
13	O	H	N	W	+
14	R	M	H	S	-

Outlook: S(unny), O(vercast), R(ainy)

Temperature: H(ot), M(edium), C(ool)

Humidity: H(igh), N(ormal), L(ow)

Wind: S(trong), W(eak)

	O	T	H	W	Play?
1	S	H	H	W	-
2	S	H	H	S	-
3	O	H	H	W	+
4	R	M	H	W	+
5	R	C	N	W	+
6	R	C	N	S	-
7	O	C	N	S	+
8	S	M	H	W	-
9	S	C	N	W	+
10	R	M	N	W	+
11	S	M	N	S	+
12	O	M	H	S	+
13	O	H	N	W	+
14	R	M	H	S	-

Example: will we play tennis today?

- We have **4 features** and produce "**look-up tables**" with the probability of playing tennis for a given attribute. As there are 9 instances of playing tennis and 5 of no tennis, we have P(Play = Yes) = 9/14 and P(Play = No) = 5/14.

OUTLOOK	Play = Yes	Play = No	Total
Sunny (S)	2/9	3/5	5/14
Overcast (O)	4/9	0/5	4/14
Rainy (R)	3/9	2/5	5/14

TEMP	Play = Yes	Play = No	Total
Hot (H)	2/9	2/5	4/14
Mild (M)	4/9	2/5	6/14
Cool (C)	3/9	1/5	4/14

HUMIDITY	Play = Yes	Play = No	Total
High (H)	3/9	4/5	7/14
Normal (N)	6/9	1/5	7/14

WIND	Play = Yes	Play = No	Total
Strong (S)	3/9	3/5	6/14
Weak (W)	6/9	2/5	8/14

Example: will we play tennis

Given a new instance x = (outlook = S, temperature = C, humidity = H, wind = S) will we play tennis or not?

OUTLOOK	Play = Yes	Play = No	Total
Sunny (S)	2/9	3/5	5/14
Overcast (O)	4/9	0/5	4/14
Rainy (R)	3/9	2/5	5/14

TEMP	Play = Yes	Play = No	Total
Hot (H)	2/9	2/5	4/14
Mild (M)	4/9	2/5	6/14
Cool (C)	3/9	1/5	4/14

HUMIDITY	Play = Yes	Play = No	Total
High (H)	3/9	4/5	7/14
Normal (N)	6/9	1/5	7/14

WIND	Play = Yes	Play = No	Total
Strong (S)	3/9	3/5	6/14
Weak (W)	6/9	2/5	8/14

- Our probabilities are

$P(S \mid Yes) = 2/9$ $P(S \mid No) = 3/5$
$P(C \mid Yes) = 3/9$ $P(C \mid No) = 1/5$
$P(H \mid Yes) = 3/9$ $P(H \mid No) = 4/5$
$P(S \mid Yes) = 3/9$ $P(S \mid No) = 3/5$

$P(x \mid Yes)\, P(Yes) = (2/9) * (3/9) * (3/9) * (3/9) * (9/14) = 0.0053$

$P(x \mid No)\, P(No) = (3/5) * (1/5) * (4/5) * (3/5) * (5/14) = 0.0206$

Example: will we play tennis

Given a new instance x = (outlook = S, temperature = C, humidity = H, wind = S) will we play tennis or not?

OUTLOOK	Play = Yes	Play = No	Total
Sunny (S)	2/9	3/5	5/14
Overcast (O)	4/9	0/5	4/14
Rainy (R)	3/9	2/5	5/14

TEMP	Play = Yes	Play = No	Total
Hot (H)	2/9	2/5	4/14
Mild (M)	4/9	2/5	6/14
Cool (C)	3/9	1/5	4/14

HUMIDITY	Play = Yes	Play = No	Total
High (H)	3/9	4/5	7/14
Normal (N)	6/9	1/5	7/14

WIND	Play = Yes	Play = No	Total
Strong (S)	3/9	3/5	6/14
Weak (W)	6/9	2/5	8/14

- Our probabilities are

P(S | Yes) = 2/9 P(S | No) = 3/5

P(C | No) = 1/5

| No) = 4/5

P(x | Yes) P(Yes) = (2/9) * (3/9) * (3/9) * (9/14) = 0.0053

P(x | No) P(No) = (3/5) * (1/5) * (4/5) * (3/5) * (5/14) = 0.0206

WE ARE NOT PLAYING TENNIS TODAY!

Parameter estimation

- As we just saw, we can **estimate distributions** of our relevant random variables **empirically using training data**. For each outcome $\tilde{x}$, we looked at the **empirical rate** of that value and determined its probability:

$$P(\tilde{x}) = \frac{\text{count}(\tilde{x})}{\text{total samples}}$$

OUTLOOK	Play = Yes
Sunny (S)	2/9
Overcast (O)	4/9
Rainy (R)	3/9

- Alternatively, we could estimate distributions via **elicitation**; we involve a human. However, this usually requires domain expertise and sophisticated ways of extracting probabilities, which are often **difficult to calibrate**.

Connection to likelihood

$$P(\tilde{x}) = \frac{\text{count}(\tilde{x})}{\text{total samples}}$$

- The empirical probability $P(\tilde{x})$ corresponds to the estimate that **maximises the likelihood** of the data (for given model parameters θ):

$$L(\theta|\tilde{x}) = \prod_l P(\tilde{x}^{(l)}|\theta) \equiv \prod_l P_\theta(\tilde{x}^{(l)})$$

OUTLOOK	Play = Yes
Sunny (S)	2/9
Overcast (O)	4/9
Rainy (R)	3/9

- **Remember**: The likelihood $L(\theta|DS)$ is the probability of a given dataset $DS = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(D)}, y^{(D)})\}$ (of length D) under a generative model $P(\mathbf{x}, y|\theta)$ with model parameters θ:

$$L(\theta|DS) = \prod_k^D P(\mathbf{x}^{(k)}, y^{(k)}|\theta)$$

Maximum likelihood estimation (MLE)

- MLE returns the parameters θ that assign largest likelihood to the data:

$$\theta_{MLE} = \text{argmax}_{\theta}\, L(\theta|DS)$$

This should remind us of the decision rule, we introduced earlier.

- This tells us that our supervised classification approach yields maximum likelihood estimates!

$$P(y = y_j) = \text{freq}(y = y_j)/D$$

$$P(x_i = \tilde{x}|y = y_j) = \text{freq}(x_i = \tilde{x}|y = y_j)/\text{freq}(y = y_j)$$

“Zero-frequency” issue in MLE

- Maximum likelihood estimation assigns all **probability mass** to events that occur in the training data.

- If a particular feature value $\tilde{\mathbf{x}}$ never occurs with a particular class label y, then the corresponding class-conditional probability vanishes, i.e., $\mathrm{P}(\tilde{\mathbf{x}}|y) = 0$. In this case, we cannot make a prediction. This is called “zero-frequency” issue.

To account for **potentially unseen events**, we need to reserve some probability mass for these instances.

Solution: Laplace (add-one) smoothing

- Let us assume that **each possible event occurs at least once**. We can account for that by **adding 1 to the frequency** of each event.

- If a feature x_i has L_i possible values $\tilde{x}$, there are L_i different events of $\mathrm{freq}(x_i = \tilde{x} | y = y_j)$ for a given class label y_j. Hence, the class-conditional probability changes as follows:

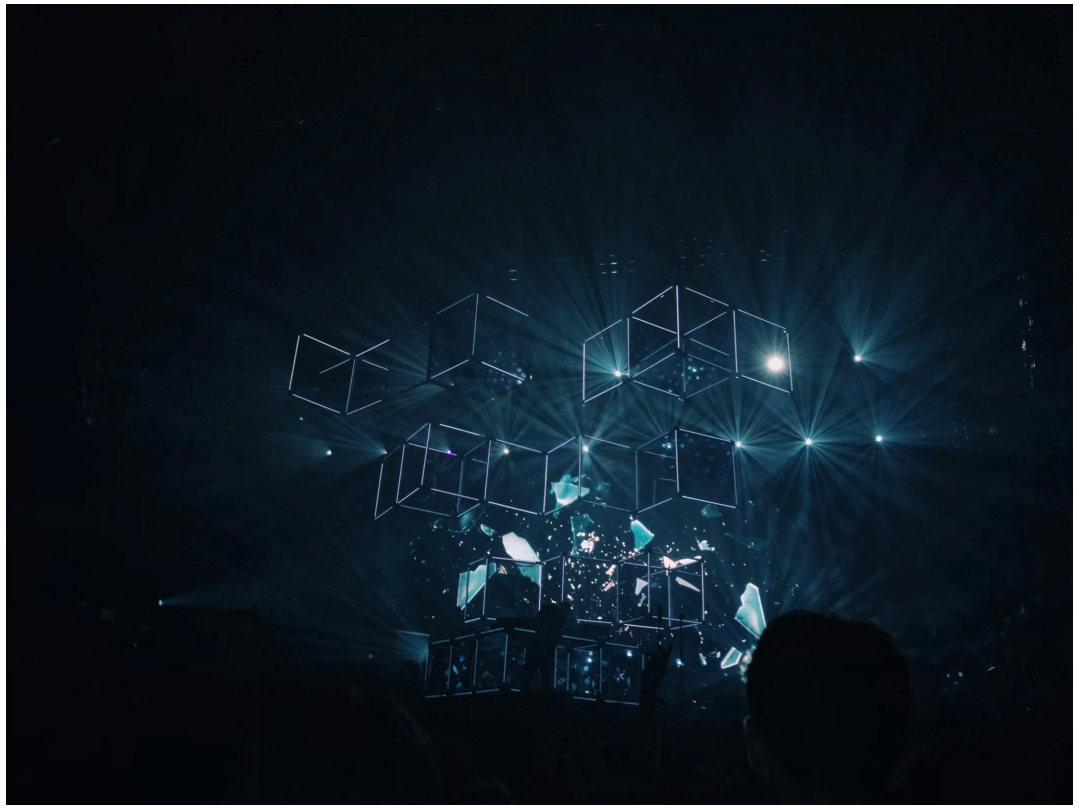
$$P_{MLE}(x_i = \tilde{x} | y = y_j) = \frac{\mathrm{freq}(x_i = \tilde{x} | y = y_j)}{\mathrm{freq}(y = y_j)}$$

OUTLOOK	Play = Yes
Sunny (S)	2/9 -> 3/12
Overcast (O)	4/9 -> 5/12
Rainy (R)	3/9 -> 4/12

$$P_{add-one}(x_i = \tilde{x} | y = y_j) = \frac{\mathrm{freq}(x_i = \tilde{x} | y = y_j) + 1}{\mathrm{freq}(y = y_j) + L_i}$$

Hypothesis space of an NB classifier

- Each set of parameters θ defines a different naïve Bayes classifier.
- Each set of parameters θ is a different hypothesis h.
- The hypothesis space encloses all models our algorithm can learns from any dataset.



Introduction

Probabilistic classification

Using a Naïve Bayes classifier

Research examples

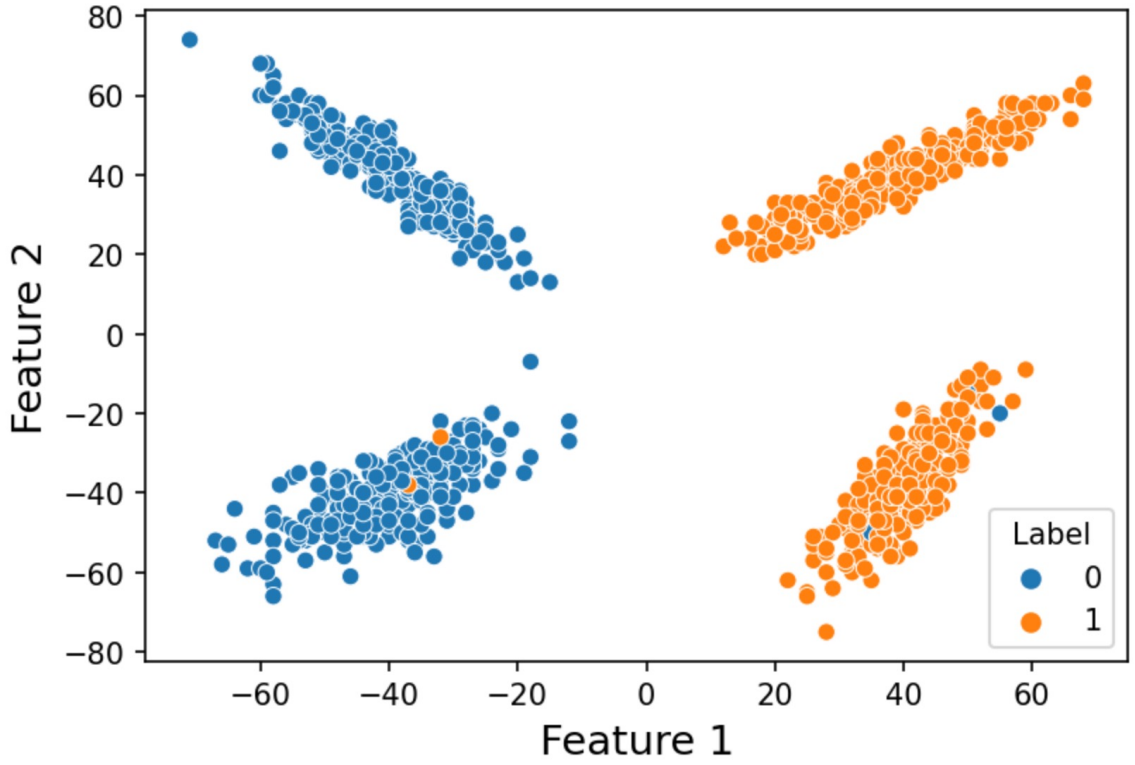
Summary

Generation of data

```
# Generate some data

x, y = make_classification(
    n_features=2, n_informative=2, n_redundant=0, class_sep=4, n_samples=1000, random_state=10
)

x[:, 0], x[:, 1] = (x[:, 0] * 10).astype(int), (x[:, 1] * 10).astype(int)
```

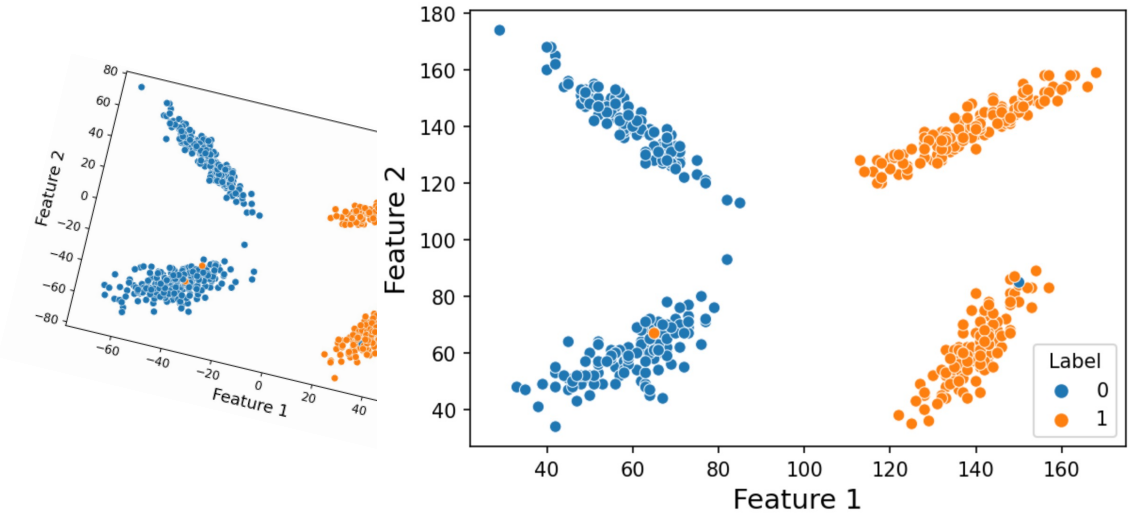
See
7_NB_classifier_examples.ipynb
for the full code for this example.

Note that there are a few incorrect class instances in some clusters.



Classification of categorical data

- Note that our values reach into the **negative range** and are numerical rather than categorical data.
- To circumvent this issue, we can **add a bias** to our data, x, to shift the values into the positive range.
- A more rigorous approach would be to pre-process the data using the LabelEncoder() feature. In this specific case, this is equivalent.




```
from sklearn.naive_bayes import CategoricalNB

clf = CategoricalNB()

# Add bias (see text and lecture)
x_train = x_train + 100
x_test = x_test + 100

clf.fit(x_train, y_train)
```

Assessing predictions: confusion matrix (recap)

- The CM summarises the **performance of a classification algorithm**. It highlights the model performance and the types of errors produced. It gives us a summary (in tabular form) of correct and incorrect predictions broken down by each category. Four outcomes exist:

True positives (TP): we predict an observation belongs to a certain class and the observation indeed belongs to that class.

False positives (FP): we predict an observation belongs to a certain class and the observation actually does NOT belong to that class (false alarm / type I error) .

False negatives (FN): we predict an observation does NOT belong to a certain class, but the observation actually belongs to that class (miss / type II error).

True negatives (TN): we predict an observation does NOT belong to a certain class, and the observation does indeed NOT belong to that class.

Confusion matrix for NB classifier

```
# use the model to predict the labels of the test data

predicted = clf.predict(x_test)
expected = y_test
```

```
# Print the confusion matrix

from sklearn import metrics

cm = metrics.confusion_matrix(expected, predicted)

print("Confusion matrix\n\n", cm)

print("\nTrue Positives(TP) = ", cm[0, 0])
print("True Negatives(TN) = ", cm[1, 1])

print("\nFalse Positives(FP) = ", cm[0, 1])
print("False Negatives(FN) = ", cm[1, 0])
```

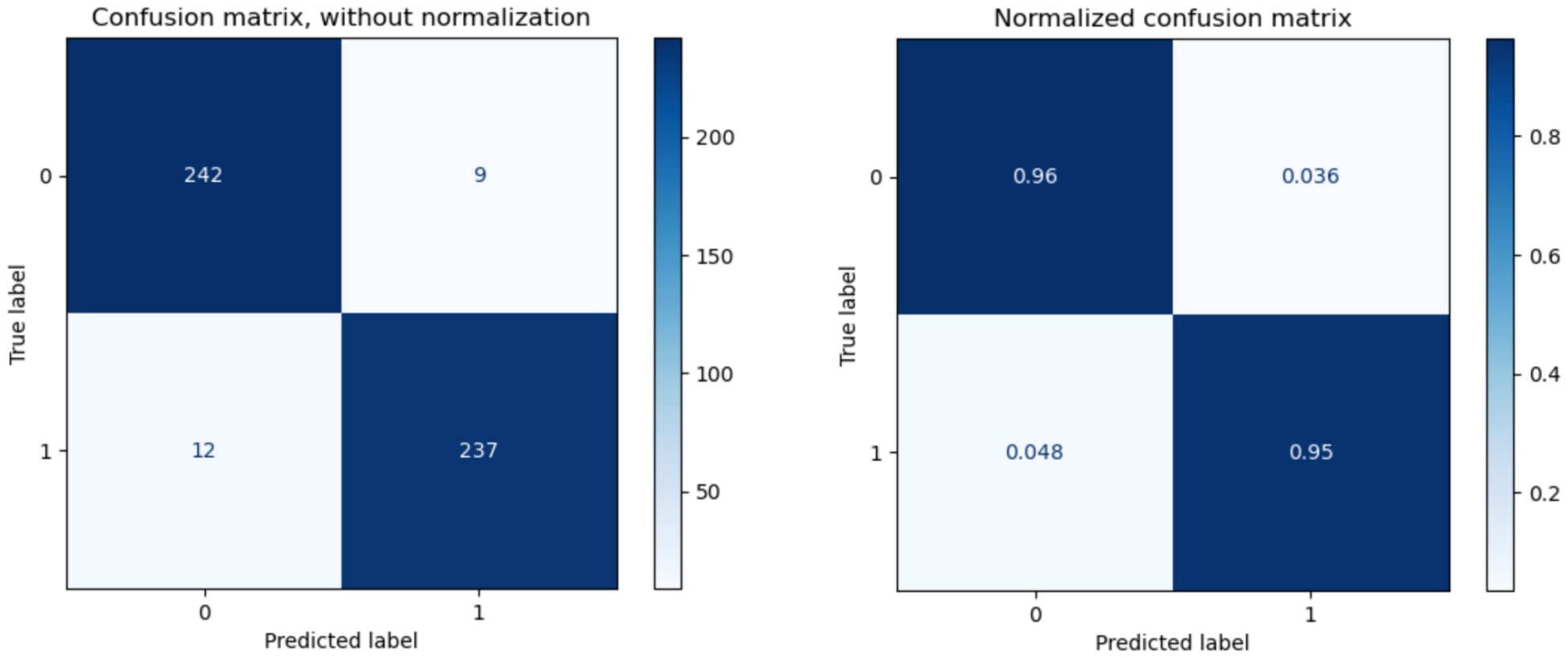
```
Confusion matrix

 [[243  24]
 [ 10 223]]

True Positives(TP) =  243
True Negatives(TN) =  223

False Positives(FP) =  24
False Negatives(FN) =  10
```

- We first **predict on our test set** and then compute the confusion matrix by **comparing to the expected labels**.

- We can output the instances for **True Positives / Negatives** along the diagonal and the **False Positives / Negatives** along the off-diagonal.

Confusion matrix for NB classifier

- Graphical representation of the confusion matrix:



Gaussian naïve Bayes

- In the previous example, we considered categorical features. What happens if they are not categorical but **continuous**? We **assume a distribution** for our class-conditional densities $P(x_i|y_j)$.

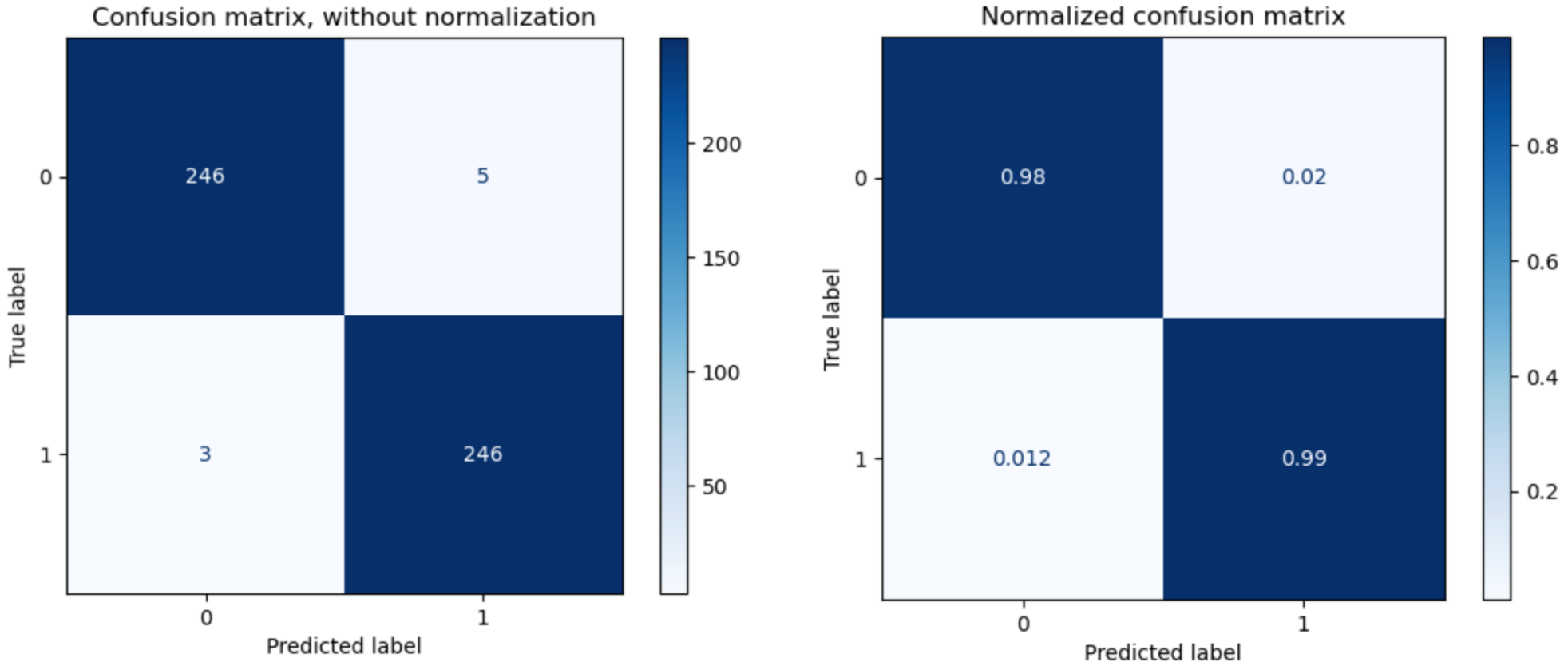
- For a **Gaussian NB** classifier, this distribution is taken to be the normal distribution. The mean and standard deviation (and the variance) of the Gaussian are specific to x_i and y_j and obtained by computing the **class-conditional means and variances** for y_j.

- Note that we do not need to add a bias in this case.

```
# Set up classifier

from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()
clf.fit(x_train, y_train)
```

Confusion matrix for Gaussian NB classifier

- Graphical representation of the confusion matrix:



Classification report

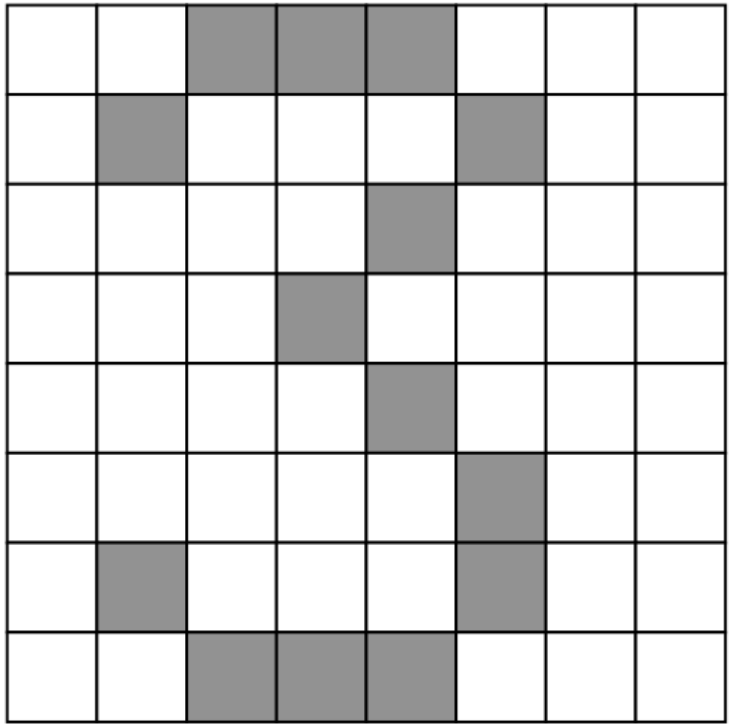
- The classification report is another way (more detailed way) to evaluate the classification model performance. It displays the precision, the recall, f1-score and support for the model.

```
print(metrics.classification_report(expected, predicted))
```

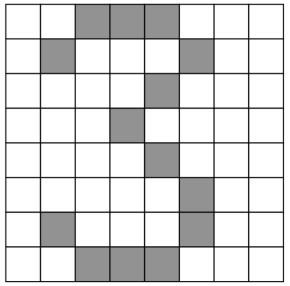
```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       251
           1       0.98      0.99      0.98       249

    accuracy                           0.98       500
   macro avg       0.98      0.98      0.98       500
weighted avg       0.98      0.98      0.98       500
```

Example: hand-written digits

- Let's focus on our **handwritten digits** again. In this case, our **input are pixel grids** (8x8) that contain a certain value. The output or label (y_k) will be a digit between 0 and 9.



Example: hand-written digits



- Let us assume the **following setup:**
  - One feature x_{ij} for each grid position <i,j>.
  - For simplicity: possible values are on (1) or off (0) based on whether the intensity of the underlying image is more or less than 0.5.
  - Each input maps to a feature vector. E.g.,

$$< x_{00} = 0, x_{01} = 0 , x_{02} = 1, , x_{03} = 1, \ldots, x_{77} = 0>$$

We have 64 features. Each has a binary value.

- We next apply our naïve Bayes' model:

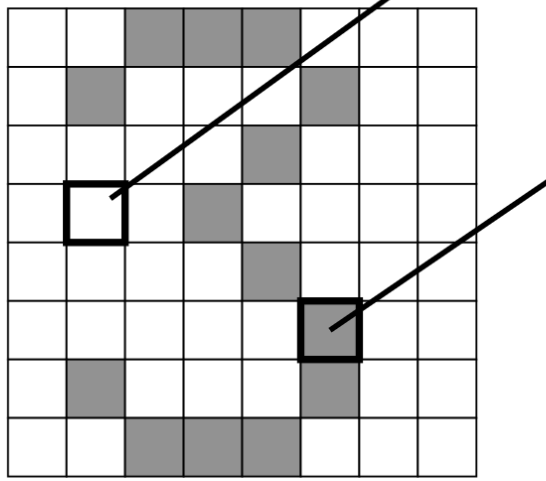
$$P(y_k|x_{00}, x_{01}, \ldots, x_{77}) \propto \prod_{i,j=1} P(x_{ij}|y_k)P(y_k)$$

Example: hand-written digits

- What do we actually need to learn?

$P(y_k)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



$P(x_{31} = on|y_k)$

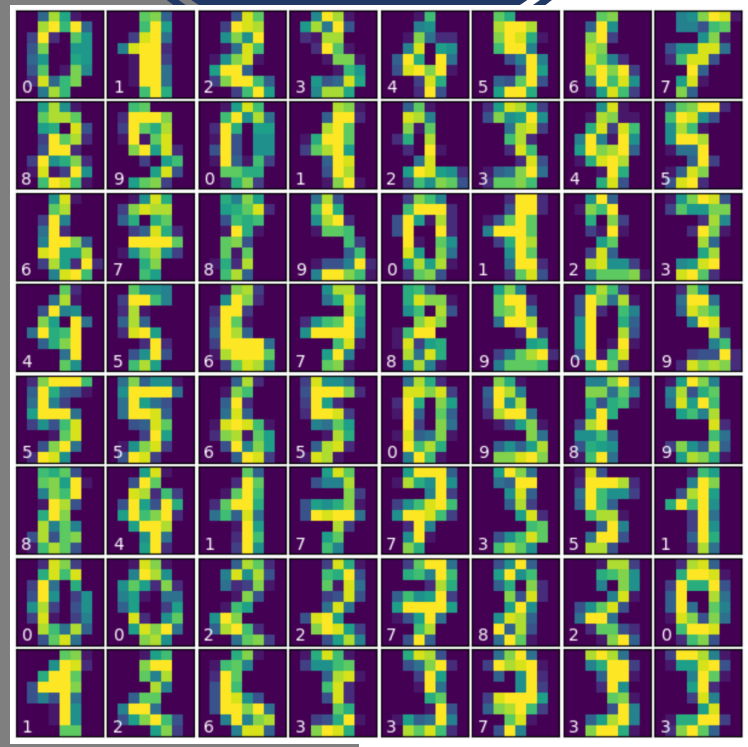
1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

$P(x_{55} = on|y_k)$

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

Our conditional probabilities encode how often in our training data, we observe a given class label (digit) if the features in a specific pixel are turned on or off.

Example: MNIST dataset



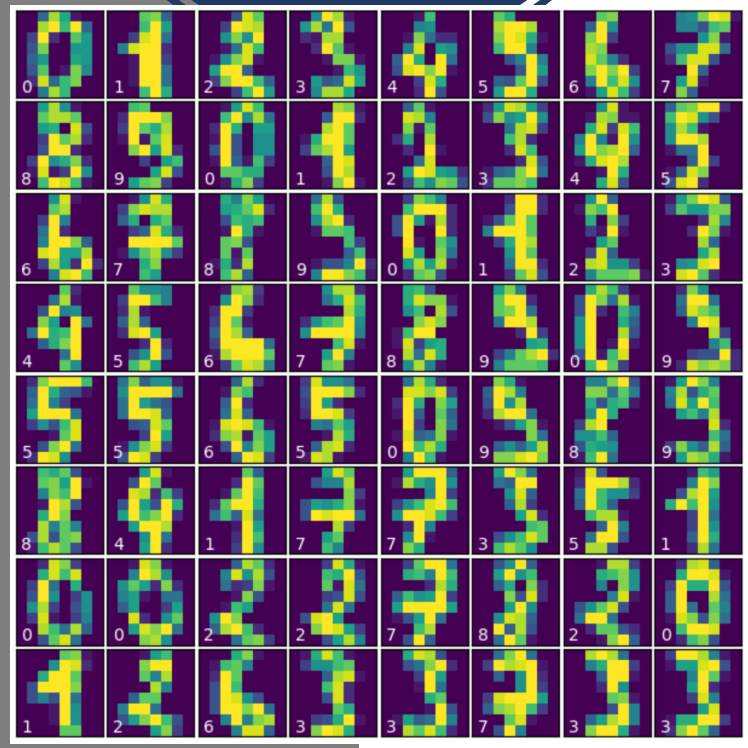
- Let's return to the MNIST dataset from earlier:

```
from sklearn.datasets import load_digits

digits = load_digits()
```

```
from matplotlib import pyplot as plt

fig = plt.figure(figsize=(6, 6))  # figure size in inches
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks=[])
    ax.imshow(digits.images[i], interpolation="nearest")
    # label the image with the target value
    ax.text(0, 7, str(digits.target[i]), color="w")
```

See 7_NB_classifier_MNIST.ipynb for the full code for this example.

Example: MNIST dataset

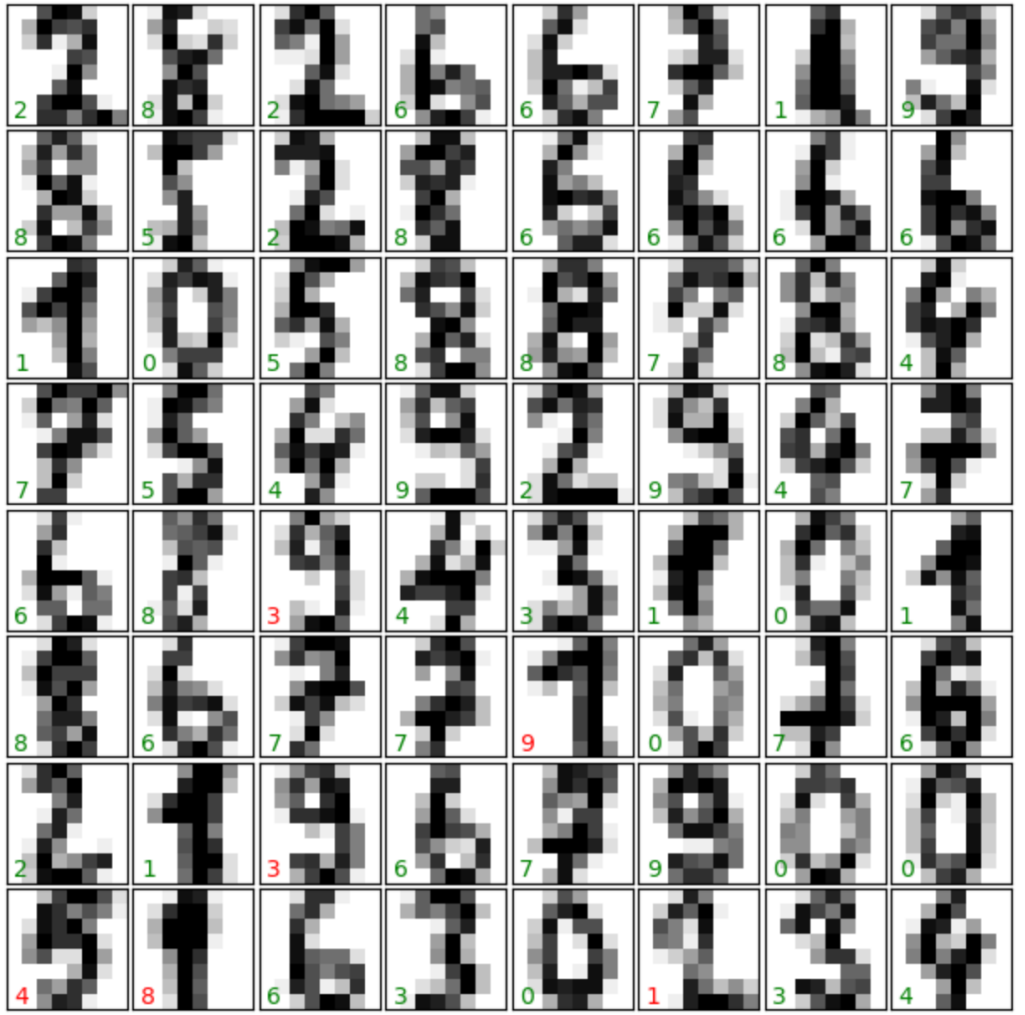


- We want to train a Gaussian NB classifier to recover the digits from our images:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# Split the data into training and validation sets

x_train, x_test, y_train, y_test = train_test_split(
    digits.data, digits.target, test_size=0.5, random_state=0
)

# Create a classifier instance and train the model

clf = GaussianNB()
clf.fit(x_train, y_train)

# Use the model to predict the labels of the test data

predicted = clf.predict(x_test)
expected = y_test
```

Example: MNIST dataset

- For test set predictions, we label images according to whether we predict digits correctly (green) or not (red):




```
# Label the image with the target value
if predicted[i] == expected[i]:
    ax.text(0, 7, str(predicted[i]), color="green")
else:
    ax.text(0, 7, str(predicted[i]), color="red")
```

Example: MNIST dataset

- To assess **performance** on the test set, we look at the **classification report**. The **averaged f1-score** is a convenient measure to analyse the overall performance.

```
# Print matches
matches = predicted == expected
print(matches.sum())

# Total number of test data points
print(len(matches))

# Fraction
print(matches.sum() / len(matches))
```

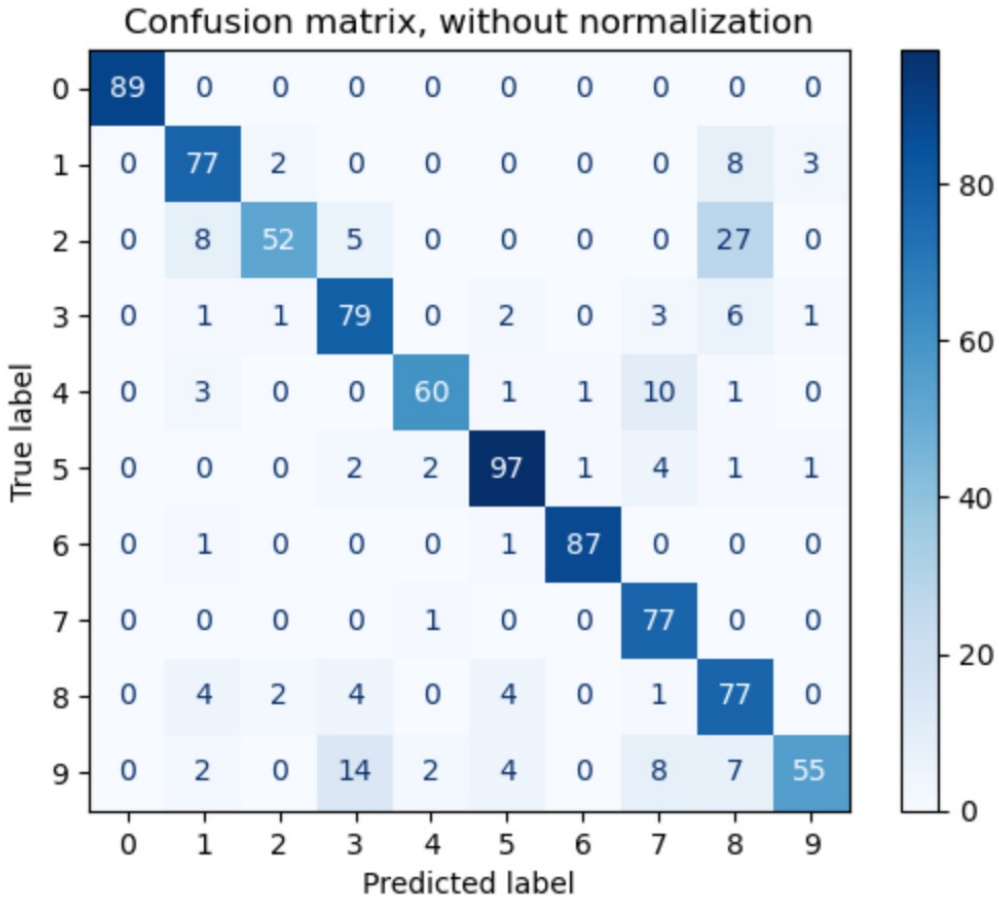
```
750
899
0.8342602892102335
```

```
from sklearn import metrics

print(metrics.classification_report(expected, predicted))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        89
           1       0.80      0.86      0.83        90
           2       0.91      0.57      0.70        92
           3       0.76      0.85      0.80        93
           4       0.92      0.79      0.85        76
           5       0.89      0.90      0.89       108
           6       0.98      0.98      0.98        89
           7       0.75      0.99      0.85        78
           8       0.61      0.84      0.70        92
           9       0.92      0.60      0.72        92

    accuracy                           0.83       899
   macro avg       0.85      0.84      0.83       899
weighted avg       0.85      0.83      0.83       899
```

Example: MNIST dataset

- The **confusion matrix** provides insights on the digits that are easy to identify, and which ones are instead sometimes mistaken for others:



Example: text classification

- NB classifiers can also be used to process text. Let us assume we want to assign a **sentiment label** $L_i \in \{\text{positive}(+), \text{negative}(-)\}$ to a document or text W_i.

- Imagine we have **two examples**:

W_1 = "This is an amazing product: great battery life, amazing features and it's cheap."

W_2 = "How awful. It's buggy, saps power and is way too expensive."

W_1 = "This is an amazing product: great battery life, amazing features and it's cheap."

W_2 = "How awful. It's buggy, saps power and is way too expensive."

Example: Bernoulli naïve Bayes

- We can, for example, imagine two different approaches to process the given words.
- The first one relies on **Bernoulli's distribution** and counts if words are present or not:

Assume W_i is a bit vector:

W_1= {amazing:1, awful:0 battery:1, buggy:0… }

W_2= {amazing:0, awful:1,battery:0, buggy:1…}

$P(W_i \mid L)$ is a Bernoulli distribution

$W_i \sim Bernoulli(\theta_L)$

$P(L)$ is a Bernoulli distribution: $L \sim Bernoulli(\pi)$

W_1 = "This is an amazing product: great battery life, amazing features and it's cheap."

W_2 = "How awful. It's buggy, saps power and is way too expensive."

Example: multi-nomial naïve Bayes

- The second approach relies on the multi-nomial distribution (generalisation of the bi-nomial distribution). It counts word instances:

Assume $\boldsymbol{W}_i$ is a "bag of words":

$\boldsymbol{W}_1 = \{amazing: 2, battery: 1, cheap: 1, features: 1\}$

$\boldsymbol{W}_2 = \{awful: 1, buggy: 1, expensive: 1, \ldots\}$

$P(\boldsymbol{W}_i \mid L)$ is a multinomial distribution:

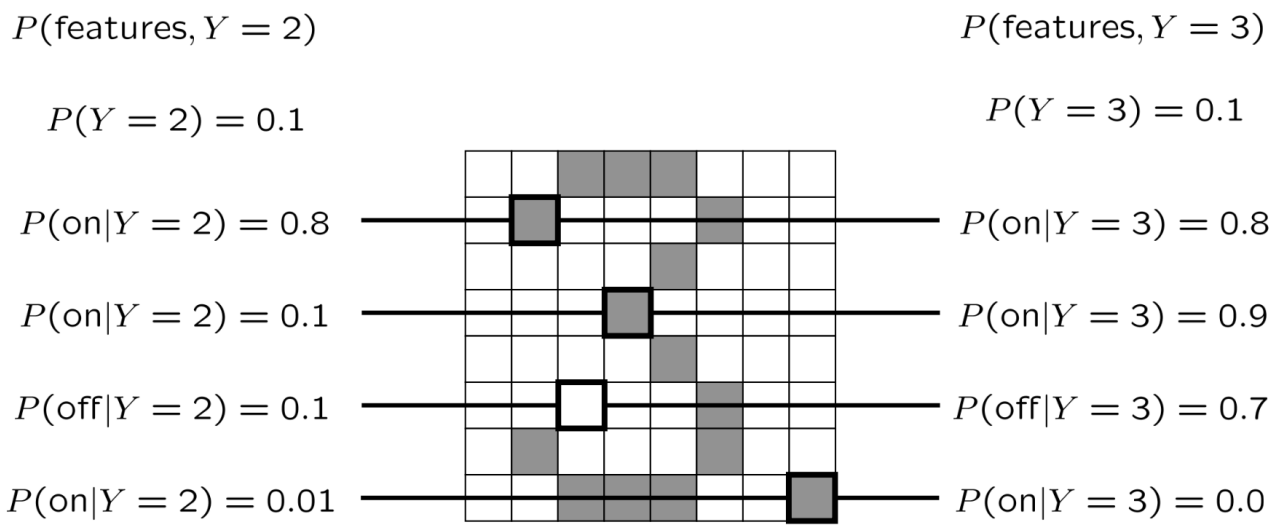
$\boldsymbol{W}_i \sim Multinomial(\boldsymbol{\theta}_L)$

We have a vocabulary of V words: $\boldsymbol{\theta}_L = (\theta_1, \ldots, \theta_V)$

$P(L)$ is a Bernoulli distribution: $L \sim Bernoulli(\pi)$

Overfitting I

- This is a key issue for ML models. The classifier performs well on training data but **does not generalise** well to unseen test data.
- Let us look at an example how this could happen for our hand-written digit problem. As before, we determine our probabilities:



Multiplying the probabilities, we find that this would be classified as a 2.

Overfitting I

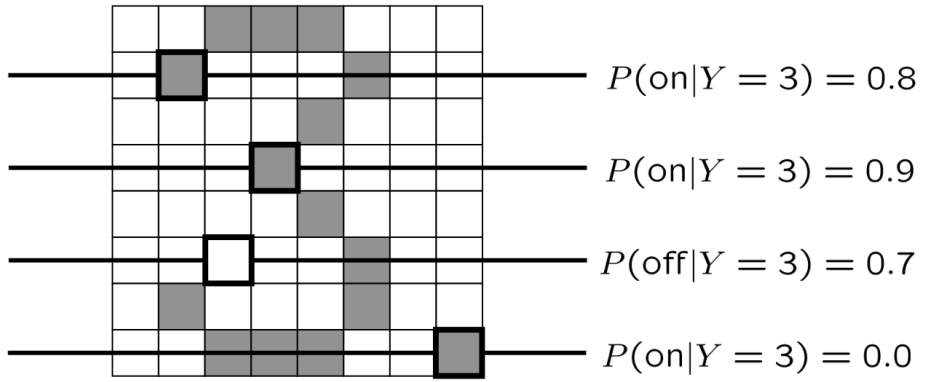
- This is a key issue for ML models. The classifier performs well on training data but **does not generalise** well to unseen test data.
- Let us look at how this could happen for our hand-written digit problem. Let us determine our probabilities:

Problematic as our posteriors are determined by relative probabilities (odds ratios)!

$P(\text{features}, Y = 2)$

$P(Y = 2) = 0.1$

$P(Y = 3) = 0.1$



...lying the ...babilities, we find that this would be classified as a 2.

Overfitting II

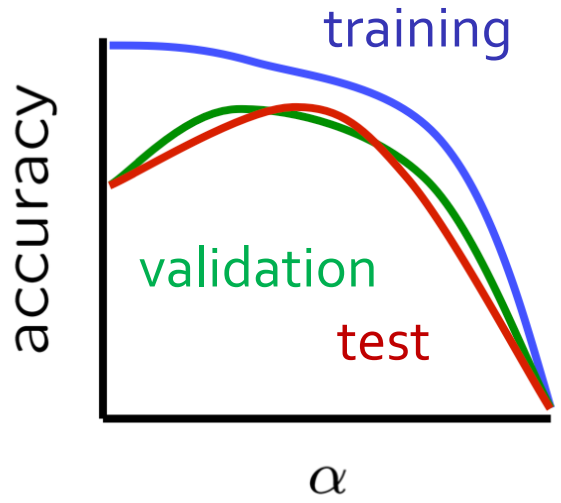
- Whenever relative frequencies are involved, we overfit training data.
  - Just because we did not see a 3 with pixel (7,7) activated during training, this does not necessarily have to hold all the time.
  - Similarly, in text recognition (e.g., spam filters) a single word such as “offer” does not always lead to spam. What about words not in the training data?
  - We cannot give unseen events zero probability.
- Imagine the extreme case of using an entire email text as the only feature. The training would result in perfect performance, while generalisation capabilities would not be available at all.

Making, e.g., a “bag-of-words” assumption helps BUT isn’t generally enough. That’s why we need to smooth or regularise our data.

Learning in practice I

- For our naïve Bayes' approach, we have **two kinds of unknowns**:
  - the parameters we want to estimate, i.e., the probabilities P(y), P(y|x).
  - the hyperparameters (e.g., amount of smoothing) we need to tune.

- How do we learn these two types of unknowns?
  - We learn the parameters from the training data.
  - We must tune the hyperparameters on different data.
  - For each set of hyperparameters, we first train on the training data and then validate on the validation set.
  - We choose the best configuration for a final test on the test dataset.



Learning in practice II

- The first step is generally to **produce a baseline** using a very simple classifier. This determines how hard the task is and establish what a "good" accuracy for the problem at hand is.

- A **weak baseline** would be one that classifies everything according to the **most frequent label**. I.e., all test labels are "ham" for a spam filter because that was most common in the training data. Accuracies could be high if the problem is skewed.

In research settings, we usually use previous works as a (strong) baseline. I.e., we want to do better than earlier studies.

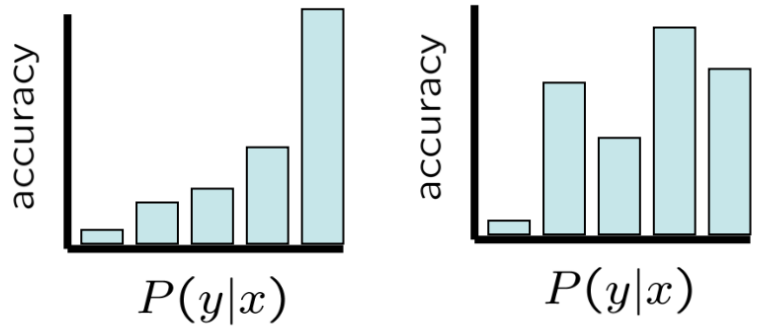
Classifier confidences

- Any probabilistic classifier (such as NB) can be assigned a confidence. It is given by the **posterior of the most probable label** y*:

$$\text{confidence}(x) = \max_y P(x|y)$$

- This information encodes how sure a classifier is of its predicted label. Note, there is no guarantee that this confidence is correct!

- To mitigate this, we can calibrate classifiers.
  - Weak calibration: higher confidence results in higher accuracy.
  - Strong calibration: confidence predicts accuracy rate.



Advantages of a NB classifier

- Now that we have discussed the theory behind a naïve Bayes classifier and seen several examples, let's summarise some key advantages:

  - It is simple and easy to implement.
  - It does not require as much training data as other classifiers.
  - It handles both continuous and discrete data.
  - It is highly scalable with number of classes and data points.
  - It is fast and can be used to make real-time predictions.
  - It is not sensitive to irrelevant features.

Disadvantages of a NB classifier

- On the other hand, some disadvantages of this approach are:

  - NB classifiers require the removal of *correlated features* as they are voted twice in the model, which over inflates their importance.
  - If a categorical variable has a category in the test set, which was not observed in the training data, then the model will assign a zero probability. I.e., the classifier is unable to make a prediction, which is known as the "zero-frequency" problem.

  - We can solve the last issue by smoothing the data. One of the simplest techniques is the Laplace (or add-one) smoothing that we saw earlier.
  - Note: sklearn applies Laplace smoothing by default for NB classifiers.

Applications

- **Real-time prediction**: NB classifiers are superfast. Thus, they can be be used for making predictions in real time such as weather forecasting.
- **Multi-class prediction:** NB classifiers allow us to predict the probability of multiple classes of target variables, not just one.
- **Text classification / spam filtering / sentiment analysis:** NB classifiers are mostly used for text classification (due to better result in multi-class problems) as they have higher success rates than other algorithms. Thus, they are widely used in spam filters or sentiment analysis (e.g., in social media settings or to identify positive/negative customer sentiments).
- **Recommendation systems:** NB classifiers are key for recommendation system to filter unseen information and predict whether a user would like a given resource or not.

Introduction

Probabilistic classification

Using a Naïve Bayes classifier

Research examples

Summary

Predicting disease

J Am Med Inform Assoc. 2011 Jul-Aug; 18(4): 370–375.
doi: 10.1136/amiajnl-2011-000101

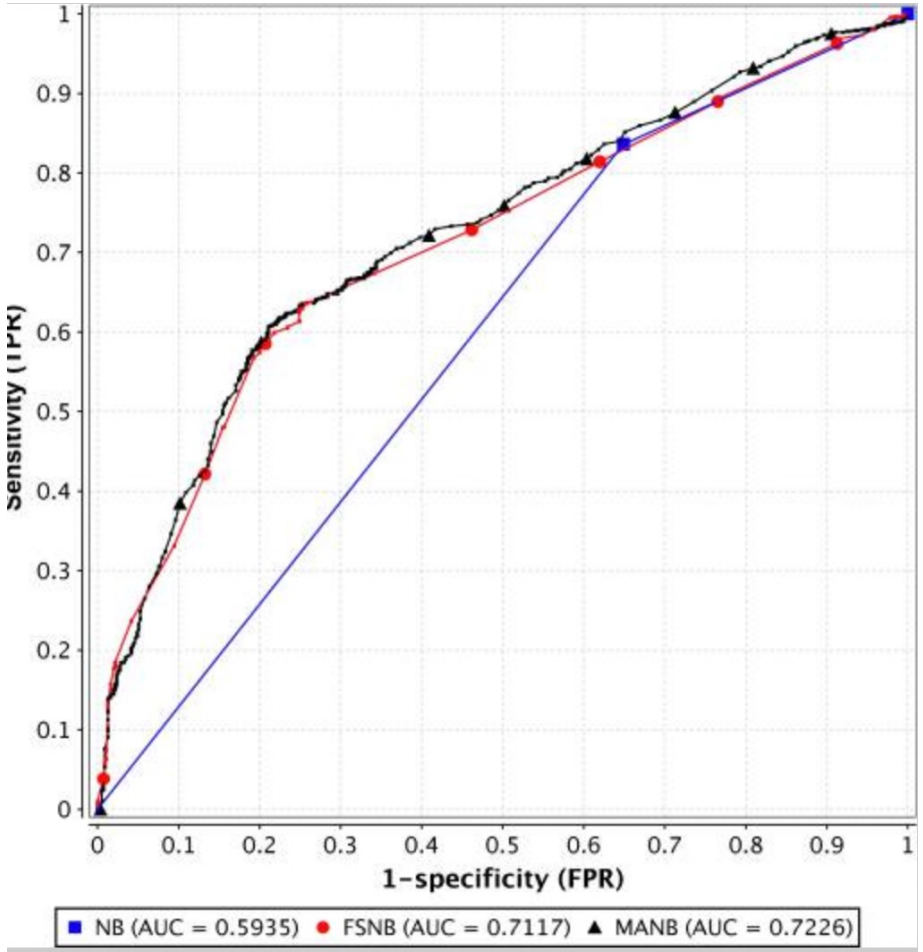
PMCID: PMC3128400
PMID: 21672907

The application of naive Bayes model averaging to predict Alzheimer's disease from genome-wide data

Wei Wei,[1] Shyam Visweswaran,[1,2] and Gregory F Cooper[1,2]

A feature-selection naive Bayes algorithm

In the experiments reported below, we also applied a version of NB that selects the features to include in the model (FSNB). In particular, we started with the model with no features. We then used a forward stepping greedy search that added the feature to the current model that most increased the score. If no additional feature increased the score, the search stopped. We scored models using the conditional marginal likelihood method described in Kontkanen *et al*,[15] combined with a binomial structure prior, which is described below. This method tries to locate the smallest set of features that predict the target variable well.



Predicting astronomical sources

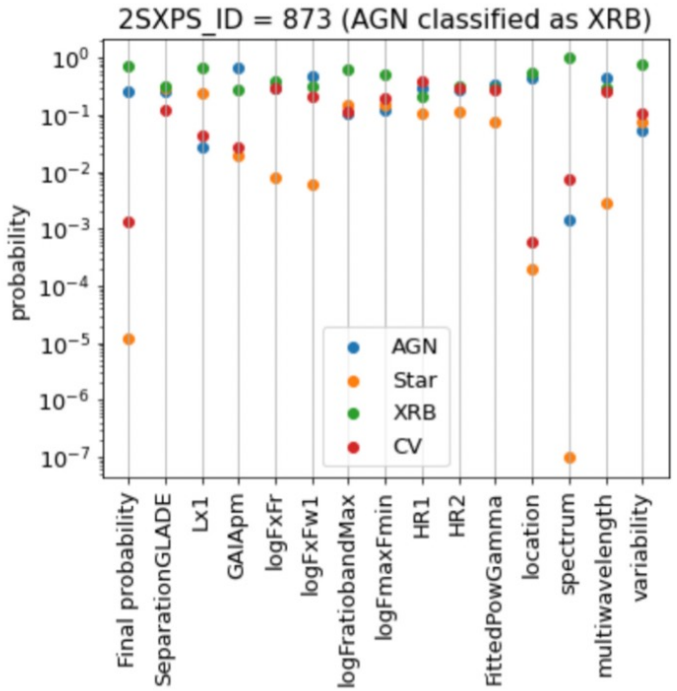
Probabilistic classification of X-ray sources applied to *Swift*-XRT and *XMM-Newton* catalogs

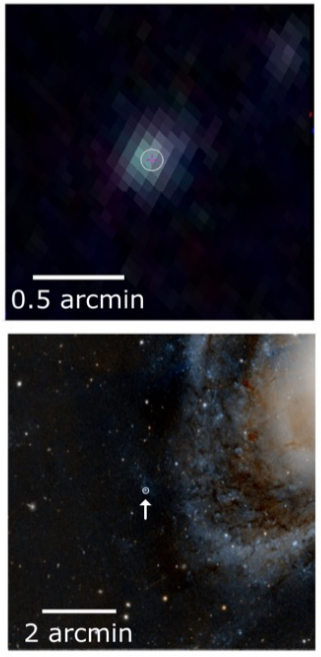
Hugo Tranin[1,2], Olivier Godet[2], Natalie Webb[2] and Daria Primorac[3,4]

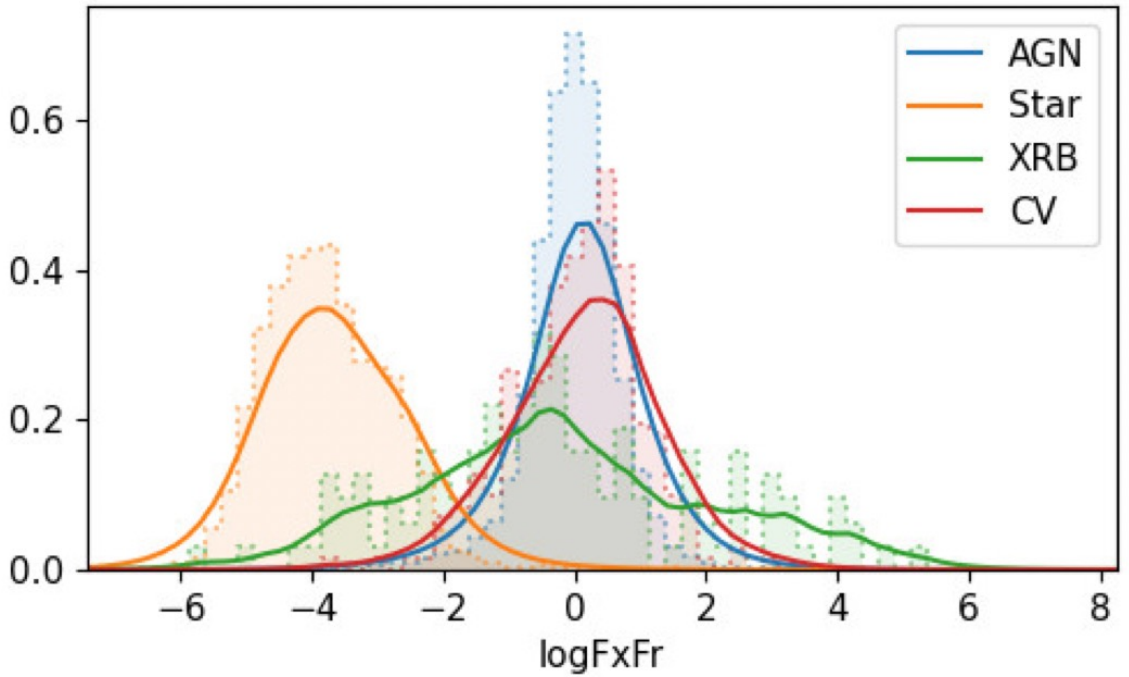
$$\mathbb{P}(\mathrm{AGN}|D) = \frac{\mathcal{P}(\mathrm{AGN})\mathcal{L}(\mathrm{AGN}|D)}{\mathcal{P}(\mathrm{AGN})\mathcal{L}(\mathrm{AGN}|D) + \mathcal{P}(Star)\mathcal{L}(Star|D)}$$

$$= \frac{0.6\ d_b^{AGN}(50°) d_{F_X/F_r}^{AGN}(-3)}{0.6\ d_b^{AGN}(50°) d_{F_X/F_r}^{AGN}(-3) + 0.4\ d_b^{Star}(50°) d_{F_X/F_r}^{Star}(-3)}$$

$$\approx 15\%,$$





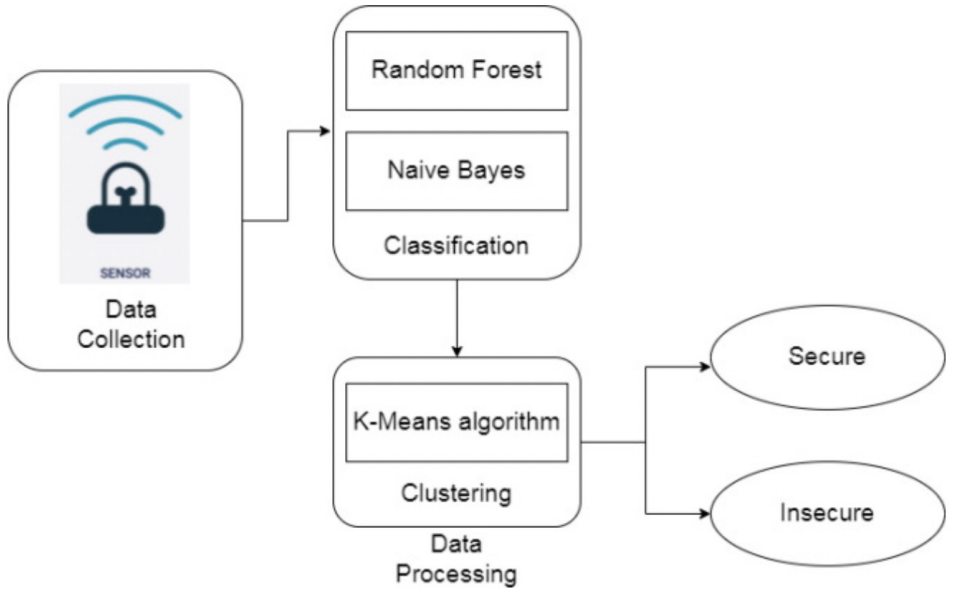


Covid-19 risk prediction

Volume 62, Part 7, 2022, Pages 4795-4799

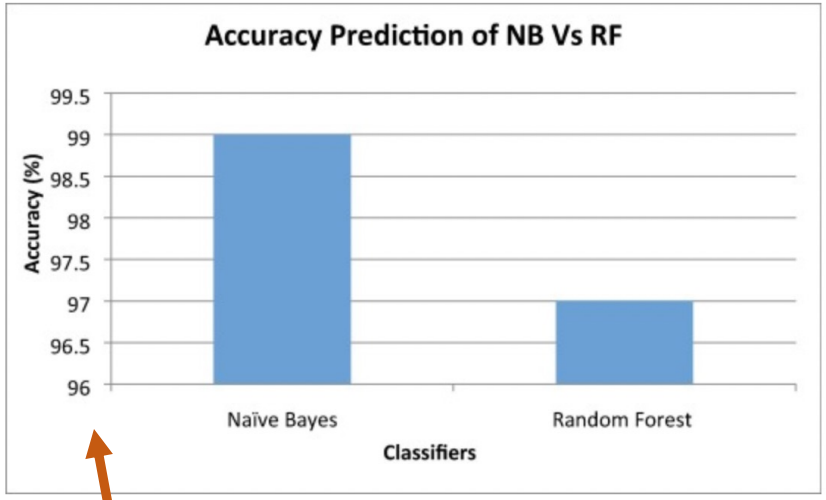
Towards applying internet of things and machine learning for the risk prediction of COVID-19 in pandemic situation using Naive Bayes classifier for improving accuracy

N. Deepa [a], J. Sathya Priya [b], Devi T. [a]



4. Results and discussion

RF algorithm and the NB algorithm are used for prediction of covid 19 diseases using parameters by the process of classification. The accuracy of classification of the former is 97% whereas the latter takes the accuracy of 99% (Fig. 2). The real time data is gathered for analyzing the performance of RF and NB.



Note: starting the y-axis at a non-zero value can be misleading. Often used in media / politics to overstate a difference.

Introduction

Probabilistic classification

Using a Naïve Bayes classifier

Research examples

Summary

Summary

- Bayes' rule lets us do diagnostic queries with causal probabilities.
- Key assumption of naïve Bayes: all features are independent given the class label.
- We can build classifiers out of a naïve Bayes' models when combined with a decision rule and training data is available.
- Due to their many advantages, they are widely used for classification tasks across a range of disciplines.