

Software

---

# Naïve Bayes

# Legal Notices and Disclaimers

This presentation is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at [intel.com](https://www.intel.com).

This sample source code is released under the [Intel Sample Source Code License Agreement](#).

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

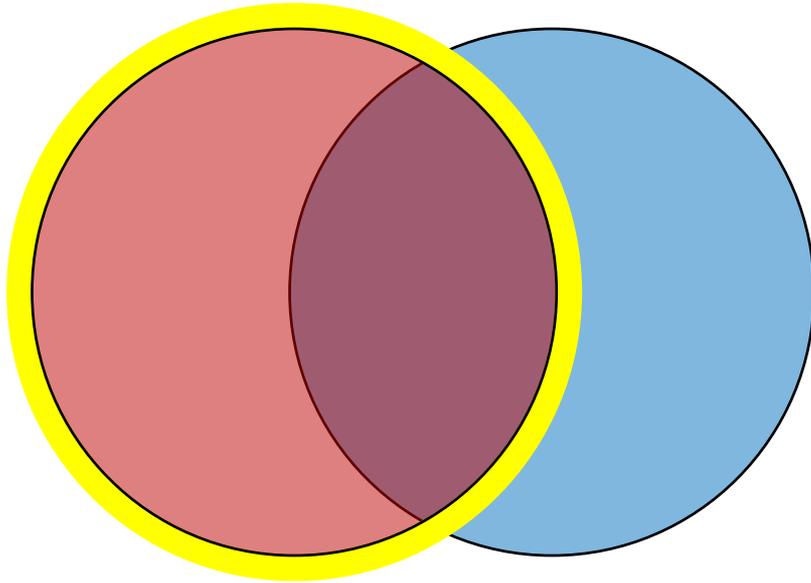
Copyright © 2021, Intel Corporation. All rights reserved.

# Learning Objectives

- Recognize basics of probability theory and its application to the Naïve Bayes classifier
- The different types of Naïve Bayes classifiers and how to train a model using this algorithm
- Apply Intel<sup>®</sup> Extension for Scikit-learn<sup>\*</sup> to leverage underlying compute capabilities of hardware

•

# Probability Basics

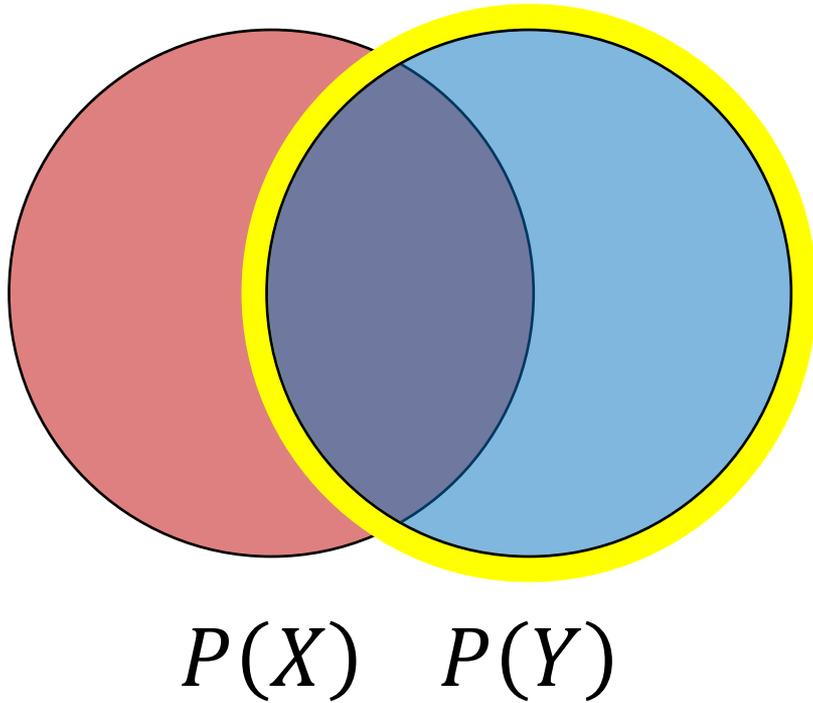


$P(X)$

- Single event probability:

$P(X)$

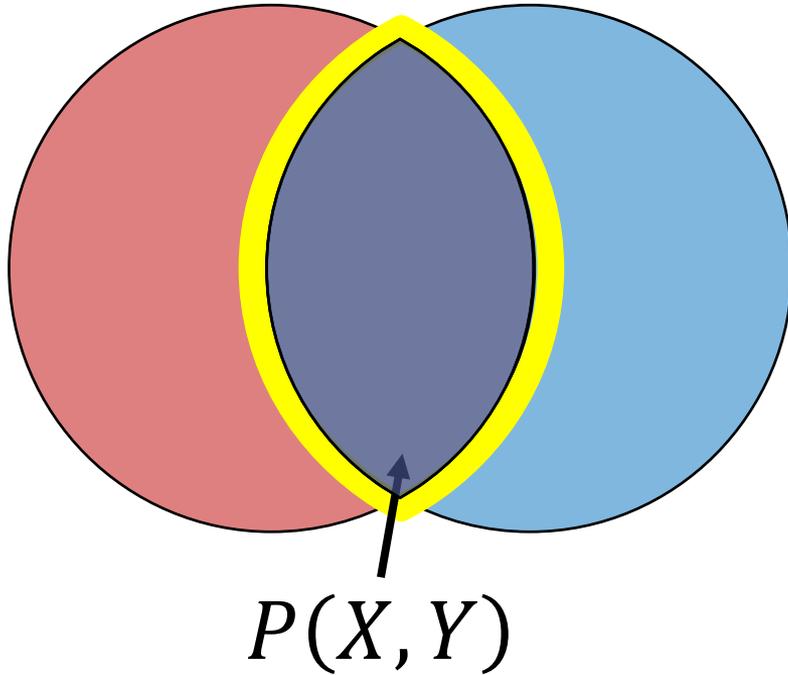
# Probability Basics



- Single event probability:

$$P(X), P(Y)$$

# Probability Basics



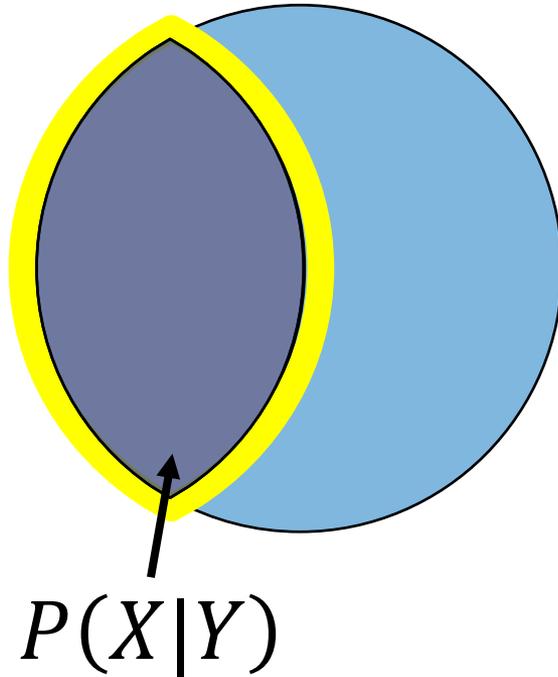
- Single event probability:

$$P(X), P(Y)$$

- Joint event probability:

$$P(X, Y)$$

# Probability Basics



- Single event probability:

$$P(X), P(Y)$$

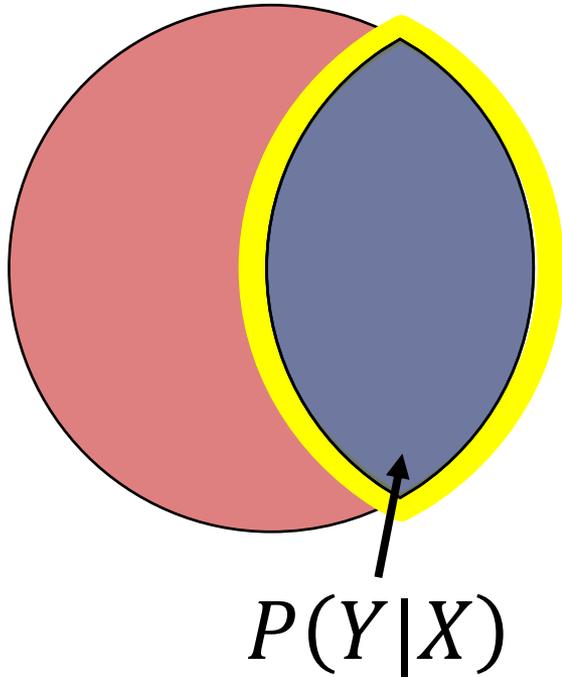
- Joint event probability:

$$P(X, Y)$$

- Conditional probability:

$$P(X|Y)$$

# Probability Basics



- Single event probability:

$$P(X), P(Y)$$

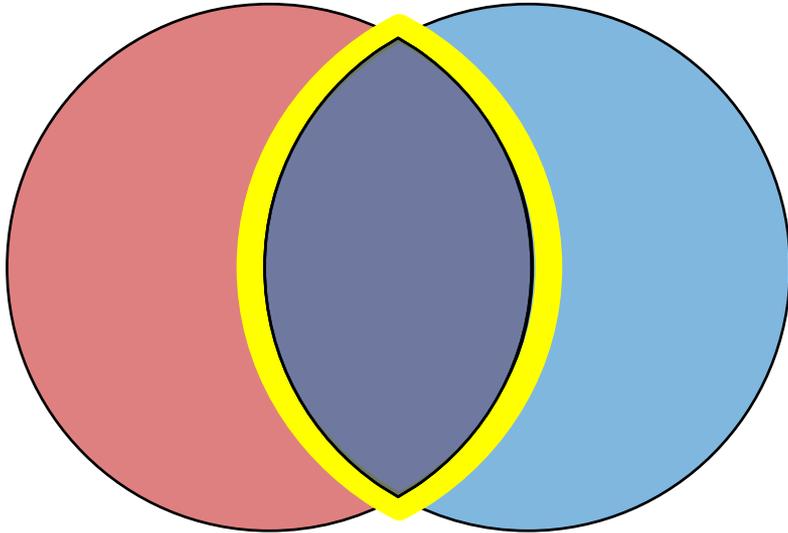
- Joint event probability:

$$P(X, Y)$$

- Conditional probability:

$$P(X|Y), P(Y|X)$$

# Probability Basics



- Single event probability:

$$P(X), P(Y)$$

- Joint event probability:

$$P(X, Y)$$

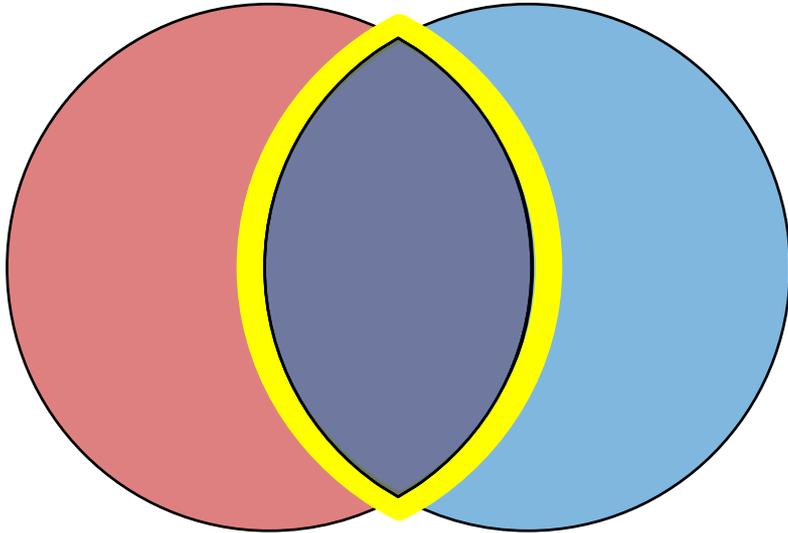
- Conditional probability:

$$P(X|Y), P(Y|X)$$

- Joint and conditional relationship:

$$P(X, Y) = P(Y|X) * P(X) = P(X|Y) * P(Y)$$

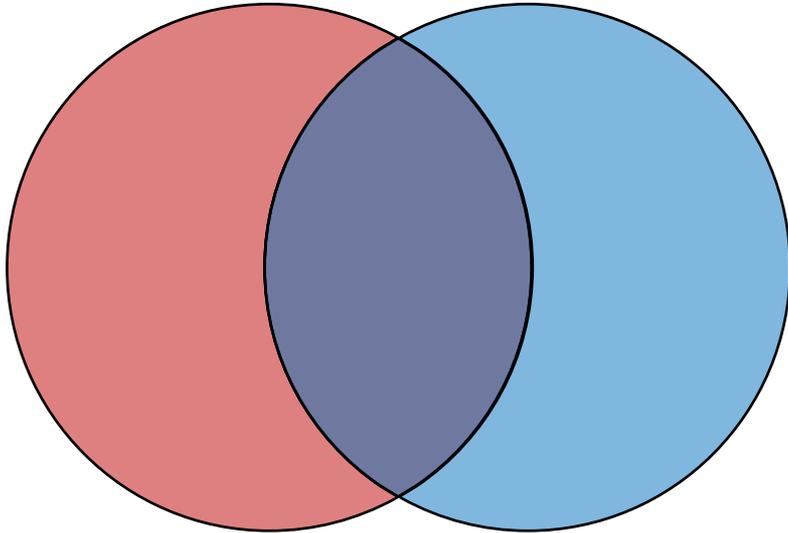
# Bayes Theorem Derivation



- By conditional and joint relationship:

$$P(Y|X) * P(X) = P(X|Y) * P(Y)$$

# Bayes Theorem Derivation



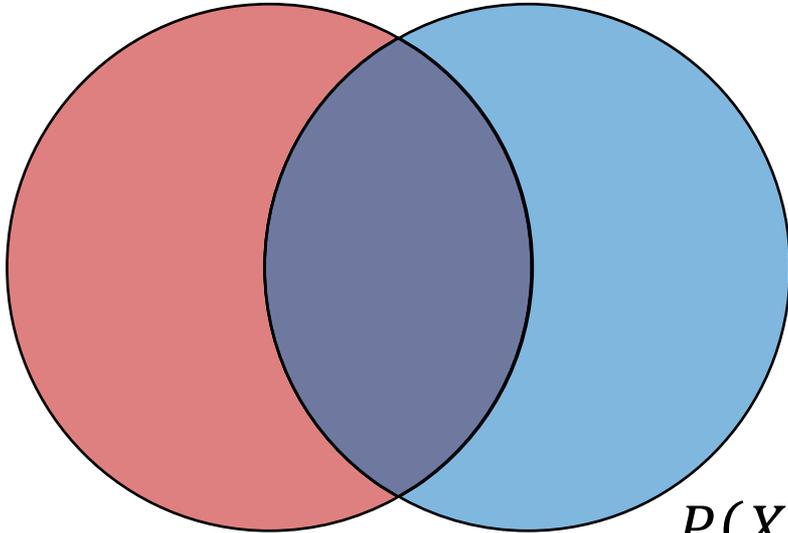
- Use conditional and joint relationship:

$$P(Y|X) * P(X) = P(X|Y) * P(Y)$$

- To invert conditional probability:

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$

# Bayes Theorem Derivation



- Use conditional and joint relationship:

$$P(Y|X) * P(X) = P(X|Y) * P(Y)$$

- To invert conditional probability:

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$

$$P(X) = \sum_Z P(X, Z) = \sum_Z P(X|Z) * P(Z)$$

# Bayes Theorem

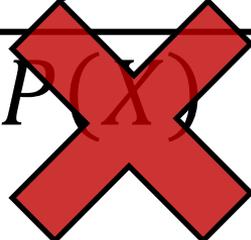
$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$

# Bayes Theorem

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$

$$\textit{posterior} = \frac{\textit{likelihood} * \textit{prior}}{\textit{evidence}}$$

# Naïve Bayes Classification

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$


$$\textit{posterior} = \frac{\textit{likelihood} * \textit{prior}}{\textit{evidence}}$$


# Training Naïve Bayes

- For each class ( $C$ ), calculate probability given features ( $X$ )

$$P(C|X) = P(X|C) * P(C)$$

**Class**   **Features**

# Training Naïve Bayes: The Naïve Assumption

- For each class ( $C$ ),  
calculate probability given  
features ( $X$ )  
$$P(C|X) = P(X|C) * P(C)$$
- Difficult to calculate joint  
probabilities produced by  
expanding for all features  
$$P(C|X) = P(X_1, X_2, \dots, X_n|C) * P(C)$$
$$P(X_1|X_2, \dots, X_n, C) * P(X_2, \dots, X_n|C) * P(C)$$
$$\dots$$

# Training Naïve Bayes: The Naïve Assumption

- For each class ( $C$ ), calculate probability given features ( $X$ )  
$$P(C|X) = P(X|C) * P(C)$$
- **Solution:** assume all features independent of each other  
$$P(C|X) = P(X_1|C) * P(X_2|C) * P(X_n|C) * P(C)$$

# Training Naïve Bayes: The Naïve Assumption

- For each class ( $C$ ), calculate probability given features ( $X$ )  
$$P(C|X) = P(X|C) * P(C)$$
- **Solution:** assume all features independent of each other  
$$P(C|X) = P(X_1|C) * P(X_2|C) * P(X_n|C) * P(C)$$
- This is the "naïve" assumption  
$$P(C|X) = P(C) \prod_{i=1}^n P(X_i|C)$$

# Training Naïve Bayes

- For each class ( $C$ ), calculate probability given features ( $X$ )
- Class assignment is selected based on *maximum a posteriori* (MAP) rule

$$P(C|X) = P(X|C) * P(C)$$

$$\underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} P(C_k) \prod_{i=1}^n P(X_i|C_k)$$

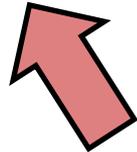
# Training Naïve Bayes

- For each class ( $C$ ), calculate probability given features ( $X$ )

$$P(C|X) = P(X|C) * P(C)$$

- Class assignment is selected based on *maximum a posteriori* (MAP) rule

$$\underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} P(C_k) \prod_{i=1}^n P(X_i|C_k)$$



Means select potential class  
with largest value

# The Log Trick

- Multiplying many values together causes computational instability (underflows)

$$\underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} P(C_k) \prod_{i=1}^n P(X_i | C_k)$$

# The Log Trick

- Multiplying many values together causes computational instability (underflows)
- Work with log values and sum the results

$$\underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} P(C_k) \prod_{i=1}^n P(X_i | C_k)$$

$$\log(P(C_k)) \sum_{i=1}^n \log(P(X_i | C_k))$$

# Example: Predicting Tennis With Naïve Bayes

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

# Example: Training Naïve Bayes Tennis Model

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Play}=\text{No}) = 5/14$$

Create probability lookup tables based on training data

# Example: Training Naïve Bayes Tennis Model

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Play}=\text{No}) = 5/14$$

<b>Outlook</b>	<b>Play=Yes</b>	<b>Play=No</b>
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

<b>Temperature</b>	<b>Play=Yes</b>	<b>Play=No</b>
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Create probability lookup tables based on training data

# Example: Training Naïve Bayes Tennis Model

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Play}=\text{No}) = 5/14$$

<b>Outlook</b>	<b>Play=Yes</b>	<b>Play=No</b>
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

<b>Humidity</b>	<b>Play=Yes</b>	<b>Play=No</b>
High	3/9	4/5
Normal	6/9	1/5

<b>Temperature</b>	<b>Play=Yes</b>	<b>Play=No</b>
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

<b>Wind</b>	<b>Play=Yes</b>	<b>Play=No</b>
Strong	3/9	3/5
Weak	6/9	2/5

Create probability lookup tables based on training data

# Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

$x' = (\text{Outlook}=\text{Sunny}, \text{Temperature}=\text{Cool}, \text{Humidity}=\text{High}, \text{Wind}=\text{Strong})$

$$P(\text{yes}|\text{sunny}, \text{cool}, \text{high}, \text{strong}) = P(\text{sunny}|\text{yes}) * P(\text{cool}|\text{yes}) * \\ P(\text{high}|\text{yes}) * P(\text{strong}|\text{yes}) * P(\text{yes})$$

$$P(\text{no}|\text{sunny}, \text{cool}, \text{high}, \text{strong}) = P(\text{sunny}|\text{no}) * P(\text{cool}|\text{no}) * \\ P(\text{high}|\text{no}) * P(\text{strong}|\text{no}) * P(\text{no})$$

# Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

$x' = (\text{Outlook}=\text{Sunny}, \text{Temperature}=\text{Cool}, \text{Humidity}=\text{High}, \text{Wind}=\text{Strong})$

<b>Feature</b>	<b>Play=Yes</b>	<b>Play=No</b>
Outlook=Sunny	2/9	3/5

# Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

$x' = (\text{Outlook}=\text{Sunny}, \text{Temperature}=\text{Cool}, \text{Humidity}=\text{High}, \text{Wind}=\text{Strong})$

Feature	Play=Yes	Play=No
Outlook=Sunny	2/9	3/5
Temperature=Cool	3/9	1/5
Humidity=High	3/9	4/5
Wind=Strong	3/9	3/5
<b>Overall Label</b>	<b>9/14</b>	<b>5/14</b>

# Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

$x' = (\text{Outlook}=\text{Sunny}, \text{Temperature}=\text{Cool}, \text{Humidity}=\text{High}, \text{Wind}=\text{Strong})$

Feature	Play=Yes	Play=No
Outlook=Sunny	2/9	3/5
Temperature=Cool	3/9	1/5
Humidity=High	3/9	4/5
Wind=Strong	3/9	3/5
<b>Overall Label</b>	<b>9/14</b>	<b>5/14</b>
<b>Probability</b>	<b>0.0053</b>	<b>0.0206</b>

# Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

$x' = (\text{Outlook}=\text{Sunny}, \text{Temperature}=\text{Cool}, \text{Humidity}=\text{High}, \text{Wind}=\text{Strong})$

Feature	Play=Yes	Play=No
Outlook=Sunny	2/9	3/5
Temperature=Cool	3/9	1/5
Humidity=High	3/9	4/5
Wind=Strong	3/9	3/5
<b>Overall Label</b>	<b>9/14</b>	<b>5/14</b>
<b>Probability</b>	<b>0.0053</b>	<b>0.0206</b>

# Laplace Smoothing

- **Problem:** categories with no entries result in a value of "0" for conditional probability

$$P(C|X) = P(X_1|C) * P(X_2|C) * P(C)$$

# Laplace Smoothing

- **Problem:** categories with no entries result in a value of "0" for conditional probability

$$P(C|X) = \overset{0}{\boxed{P(X_1|C)}} * P(X_2|C) * P(C)$$

# Laplace Smoothing

- **Problem:** categories with no entries result in a value of "0" for conditional probability
- **Solution:** add "1" to numerator and denominator of empty categories

$$P(C|X) = \overset{0}{\boxed{P(X_1|C)}} * P(X_2|C) * P(C)$$

$$P(X_1|C) = \frac{1}{\text{Count}(C) + n}$$

$$P(X_2|C) = \frac{\text{Count}(X_2 \& C) + 1}{\text{Count}(C) + m}$$

# Types of Naïve Bayes

Naïve Bayes Model

Bernoulli

Data Type

Binary (T/F)

# Types of Naïve Bayes

Naïve Bayes Model

Bernoulli

Multinomial

Data Type

Binary (T/F)

Discrete (e.g. count)

# Types of Naïve Bayes

Naïve Bayes Model

Bernoulli

Multinomial

Gaussian

Data Type

Binary (T/F)

Discrete (e.g. count)

Continuous

# Combining Feature Types

## Problem

- Model features contain different data types (continuous and categorical)

# Combining Feature Types

## Problem

- Model features contain different data types (continuous and categorical)

## Solutions

- **Option 1:** Bin continuous features to create categorical ones and fit multinomial model

# Combining Feature Types

## Problem

- Model features contain different data types (continuous and categorical)

## Solutions

- **Option 1:** Bin continuous features to create categorical ones and fit multinomial model
- **Option 2:** Fit Gaussian model on continuous features and multinomial on categorical features; combine to create "meta model" (week 10)

# Distributed Computing with Naïve Bayes

- Well-suited for large data and distributed computing—limited parameters and log probabilities are a summation
- Scikit-Learn implementations contain a "partial\_fit" method designed for out-of-core calculations

# Naïve Bayes: The Syntax

**Import the class containing the classification method**

```
from sklearn.naive_bayes import BernoulliNB
```

```
from sklearnex import
```

```
()
```

# Naïve Bayes: The Syntax

**Import the class containing the classification method**

```
from sklearn.naive_bayes import BernoulliNB
```

**To use the Intel® Extension for Scikit-learn\* variant of this algorithm:**

- **Install Intel® oneAPI AI Analytics Toolkit (AI Kit)**
- **Add the following two lines of code after the code above:**

```
from sklearnex import patch_sklearn  
patch_sklearn()
```

# Naïve Bayes: The Syntax

**Import the class containing the classification method**

```
from sklearn.naive_bayes import BernoulliNB
```

**Create an instance of the class**

```
BNB = BernoulliNB(alpha=1.0)
```

# Naïve Bayes: The Syntax

Import the class containing the classification method

```
from sklearn.naive_bayes import BernoulliNB
```

Create an instance of the class

```
BNB = BernoulliNB(alpha=1.0)
```



Laplace smoothing  
parameter

# Naïve Bayes: The Syntax

**Import the class containing the classification method**

```
from sklearn.naive_bayes import BernoulliNB
```

**Create an instance of the class**

```
BNB = BernoulliNB(alpha=1.0)
```

**Fit the instance on the data and then predict the expected value**

```
BNB = BNB.fit(X_train, y_train)
```

```
y_predict = BNB.predict(X_test)
```

# Naïve Bayes: The Syntax

**Import the class containing the classification method**

```
from sklearn.naive_bayes import BernoulliNB
```

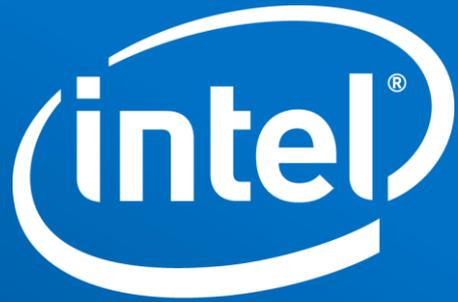
**Create an instance of the class**

```
BNB = BernoulliNB(alpha=1.0)
```

**Fit the instance on the data and then predict the expected value**

```
BNB = BNB.fit(X_train, y_train)  
y_predict = BNB.predict(X_test)
```

**Other naïve Bayes models: [MultinomialNB](#), [GaussianNB](#).**



Software

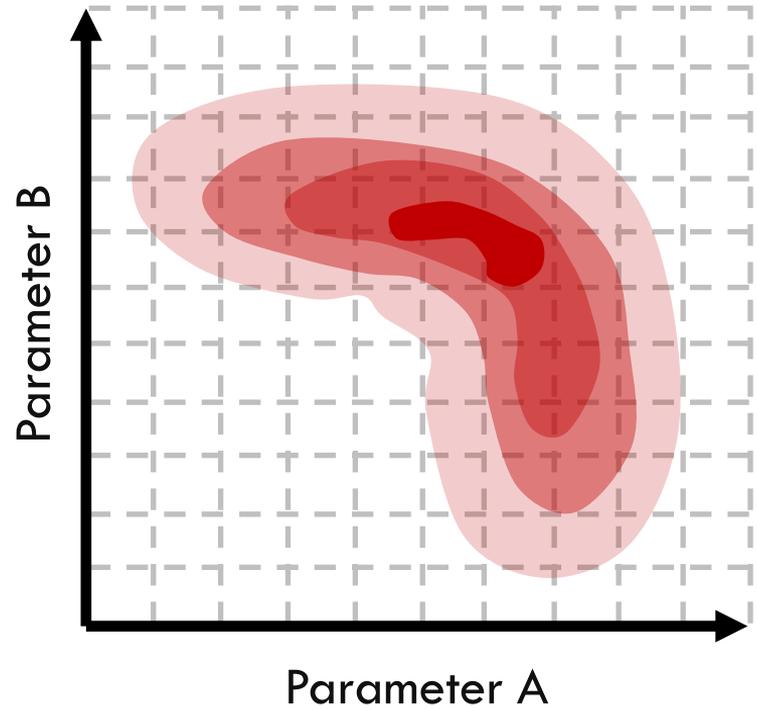


Software

# Grid Search --- and Pipelines

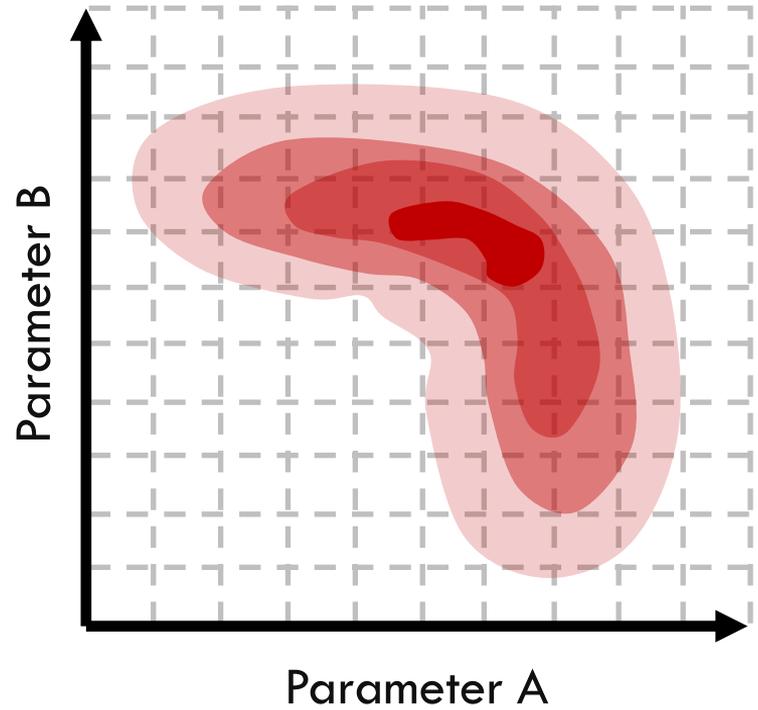
# Generalized Hyperparameter Grid Search

- Hyperparameter selection for regularization / better models requires cross validation on training data
- Linear and logistic regression methods have classes devoted to grid search (e.g. LassoCV)



# Generalized Hyperparameter Grid Search

- Grid search can be useful for other methods too, so a generalized method is desirable
- Scikit-learn contains `GridSearchCV`, which performs a grid search with parameters using cross validation



# Grid Search with Cross Validation: The Syntax

**Import the class containing the grid search method**

```
from sklearn.linear_model import LogisticRegression  
from sklearn.model_selection import GridSearchCV
```

**To use the Intel® Extension for Scikit-learn\* variant of this algorithm:**

- **Install Intel® oneAPI AI Analytics Toolkit (AI Kit)**
- **Add the following two lines of code after the code above:**

```
from sklearnex import patch_sklearn  
patch_sklearn()
```

# Grid Search with Cross Validation: The Syntax

**Import the class containing the grid search method**

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import GridSearchCV
```

# Grid Search with Cross Validation: The Syntax

**Import the class containing the grid search method**

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import GridSearchCV
```

# Grid Search with Cross Validation: The Syntax

**Import the class containing the grid search method**

```
from sklearn.linear_model import LogisticRegression  
from sklearn.model_selection import GridSearchCV
```

**Create an instance of the estimator and grid search class**

```
LR = LogisticRegression(penalty='l2')  
GS = GridSearchCV(LR, param_grid={'c':[0.001, 0.01, 0.1]},  
                    scoring='accuracy', cv=4)
```

# Grid Search with Cross Validation: The Syntax

Import the class containing the grid search method

```
from sklearn.linear_model import LogisticRegression  
from sklearn.model_selection import GridSearchCV
```

Create an instance of the estimator and grid search class

```
LR = LogisticRegression(penalty='l2')  
GS = GridSearchCV(LR, param_grid={'c':[0.001, 0.01, 0.1]},  
                    scoring='accuracy', cv=4)
```

logistic regression  
method



# Grid Search with Cross Validation: The Syntax

**Import the class containing the grid search method**

```
from sklearn.linear_model import LogisticRegression  
from sklearn.model_selection import GridSearchCV
```

**Create an instance of the estimator and grid search class**

```
LR = LogisticRegression(penalty='l2')  
GS = GridSearchCV(LR, param_grid={'c':[0.001, 0.01, 0.1]},  
                    scoring='accuracy', cv=4)
```

**Fit the instance on the data to find the best model and then predict**

```
GS = GS.fit(X_train, y_train)  
y_train = GS.predict(X_test)
```

# Optimizing the Rest of the Pipeline

- Grid searches enable model parameters to be optimized

# Optimizing the Rest of the Pipeline

- Grid searches enable model parameters to be optimized
- How can this be incorporated with other steps of the process (e.g. feature extraction and transformation)?

# Optimizing the Rest of the Pipeline

- Grid searches enable model parameters to be optimized
- How can this be incorporated with other steps of the process (e.g. feature extraction and transformation)?



Pipelines!

# Automating Machine Learning with Pipelines

- Machine learning models often selected empirically



# Automating Machine Learning with Pipelines

- Machine learning models often selected empirically
- By trying different processing methods and tuning multiple models



# Automating Machine Learning with Pipelines

- Machine learning models often selected empirically
- By trying different processing methods and tuning multiple models



How to automate this process?

# Automating Machine Learning with Pipelines

- Pipelines in Scikit-Learn allow feature transformation steps and models to be chained together



# Automating Machine Learning with Pipelines

- Pipelines in Scikit-Learn allow feature transformation steps and models to be chained together
- Successive steps perform 'fit' and 'transform' before sending data to the next step



# Automating Machine Learning with Pipelines

- Pipelines in Scikit-Learn allow feature transformation steps and models to be chained together
- Successive steps perform 'fit' and 'transform' before sending data to the next step



Pipelines make automation and reproducibility easier!

# Pipelines: The Syntax

**Import the class containing the pipeline method**

```
from sklearn.pipeline import Pipeline
```

# Pipelines: The Syntax

**Import the class containing the pipeline method**

```
from sklearn.pipeline import Pipeline
```

**Create an instance of the class with estimators**

```
estimators = [('scaler', MinMaxScaler()), ('lasso', Lasso())]
```

```
Pipe = Pipeline(estimators)
```

# Pipelines: The Syntax

Import the class containing the pipeline method

```
from sklearn.pipeline import Pipeline
```

Create an instance of the class with estimators

```
estimators = [('scaler', MinMaxScaler()), ('lasso', Lasso())]
```

```
Pipe = Pipeline(estimators)
```

feature scaler  
class



# Pipelines: The Syntax

Import the class containing the pipeline method

```
from sklearn.pipeline import Pipeline
```

Create an instance of the class with estimators

```
estimators = [('scaler', MinMaxScaler()), ('lasso', Lasso())]
```

```
Pipe = Pipeline(estimators)
```



# Pipelines: The Syntax

Import the class containing the pipeline method

```
from sklearn.pipeline import Pipeline
```

Create an instance of the class with estimators

```
estimators = [('scaler', MinMaxScaler()), ('lasso', Lasso())]
```

```
Pipe = Pipeline(estimators)
```

Fit the instance on the data and then predict the expected value

```
Pipe = Pipe.fit(X_train, y_train)
```

```
y_predict = Pipe.predict(X_test)
```

# Pipelines: The Syntax

Import the class containing the pipeline method

```
from sklearn.pipeline import Pipeline
```

Create an instance of the class with estimators

```
estimators = [('scaler', MinMaxScaler()), ('lasso', Lasso())]
```

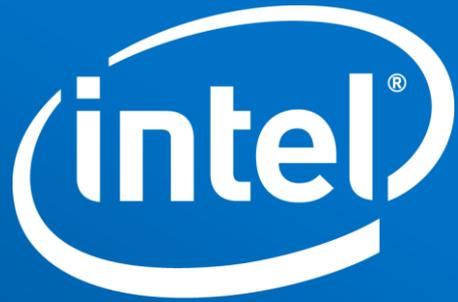
```
Pipe = Pipeline(estimators)
```

Fit the instance on the data and then predict the expected value

```
Pipe = Pipe.fit(X_train, y_train)
```

```
y_predict = Pipe.predict(X_test)
```

Features can be combined from different transform method using **FeatureUnion**



Software