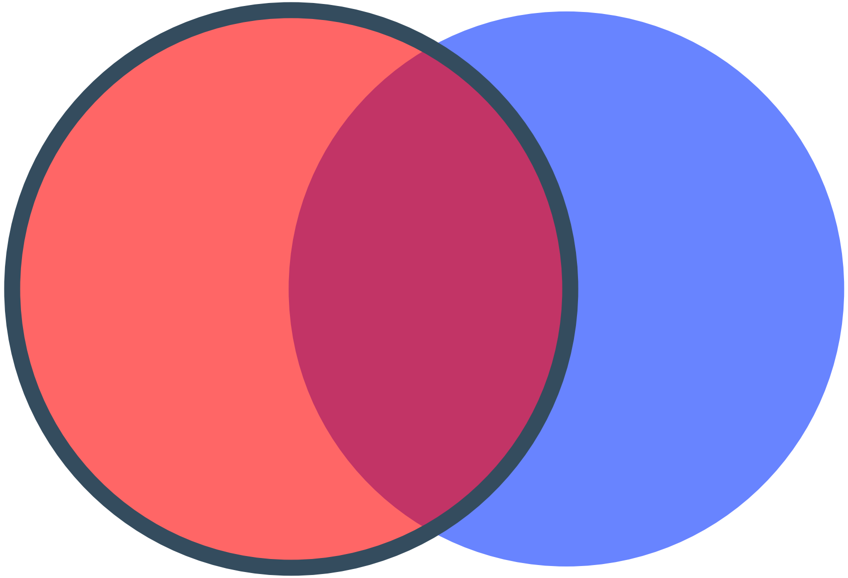
Intel Nervana AI Academy

NAÏVE BAYES

PROBABILITY BASICS

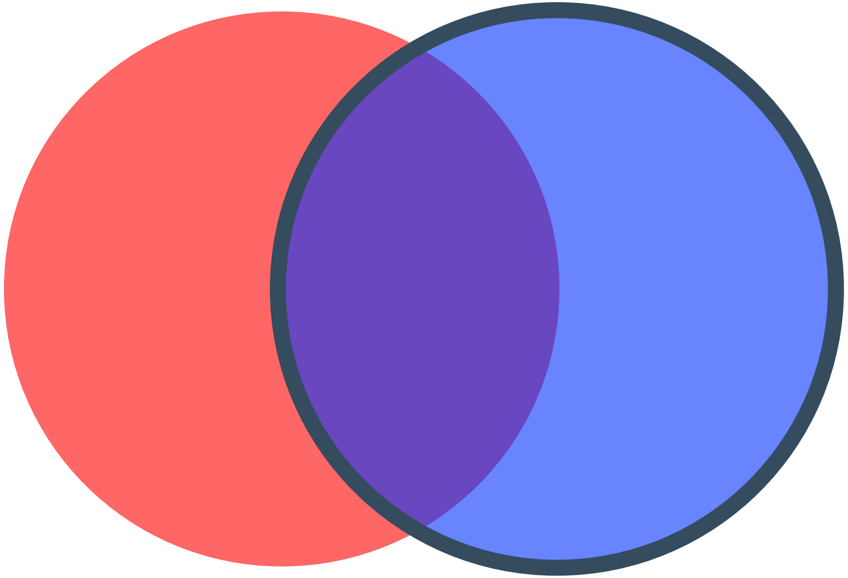


$P(X)$

LinSingle event probability:

$$P(X)$$

PROBABILITY BASICS

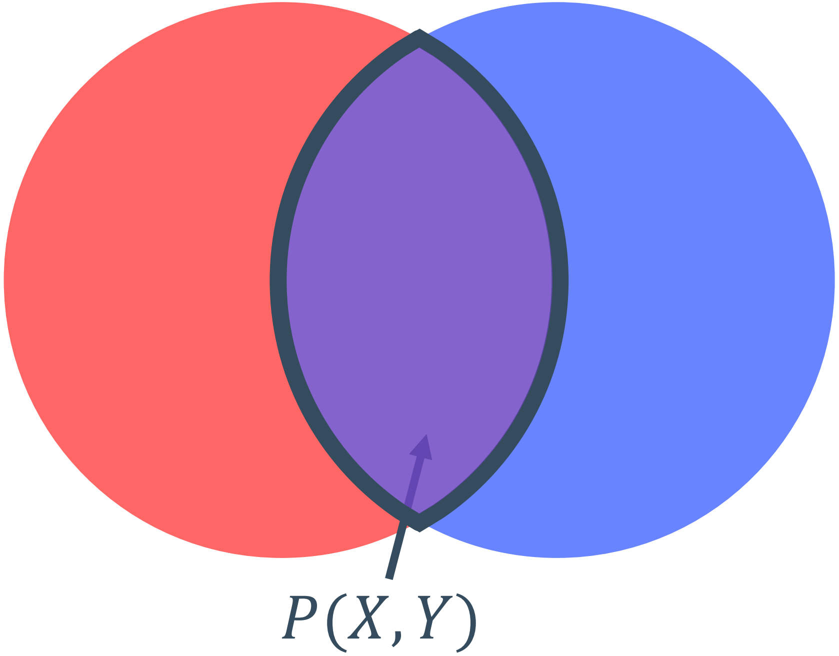


$P(Y)$

LinSingle event probability:

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

PROBABILITY BASICS



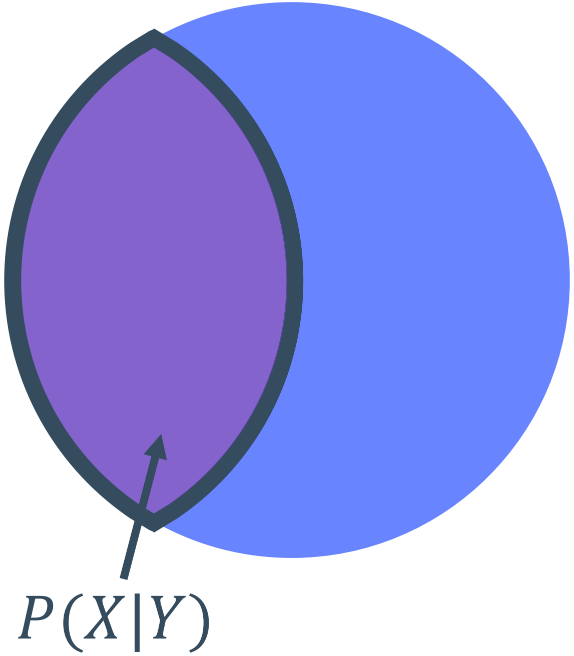
LinSingle event probability:

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

Joint event probability:

$$P(X,Y)$$

PROBABILITY BASICS



LinSingle event probability:

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

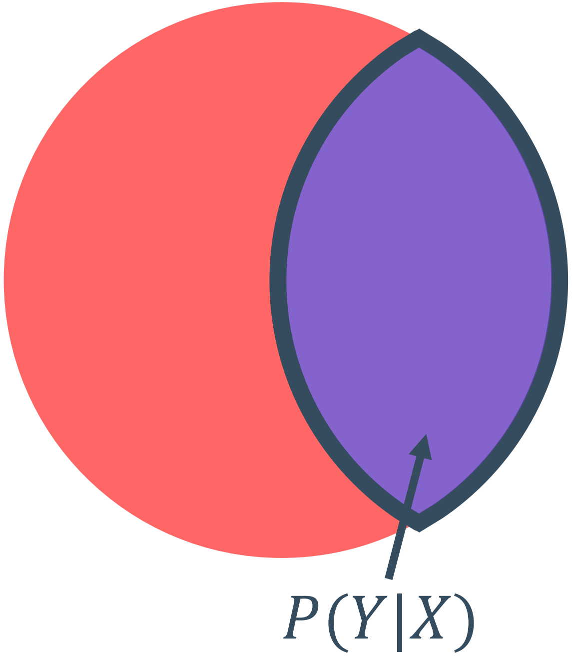
Joint event probability:

$$P(X, Y)$$

Conditional probability:

$$P(X|Y)$$

PROBABILITY BASICS



LinSingle event probability:

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

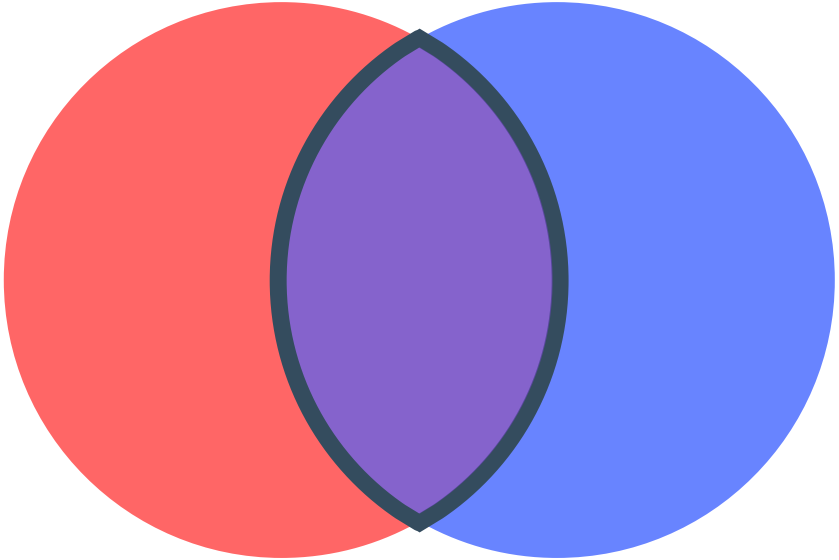
Joint event probability:

$$P(X, Y)$$

Conditional probability:

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

PROBABILITY BASICS



LinSingle event probability:

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

Joint event probability:

$$P(X, Y)$$

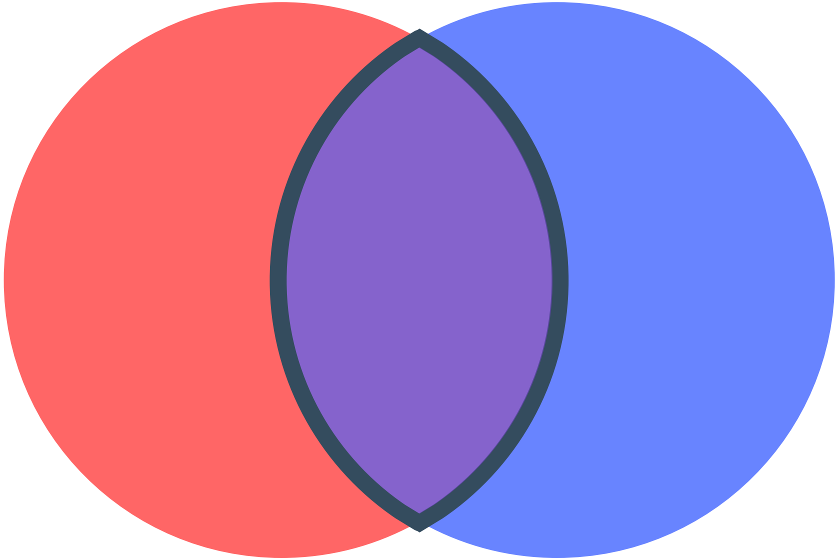
Conditional probability:

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

Joint and conditional relationship:

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

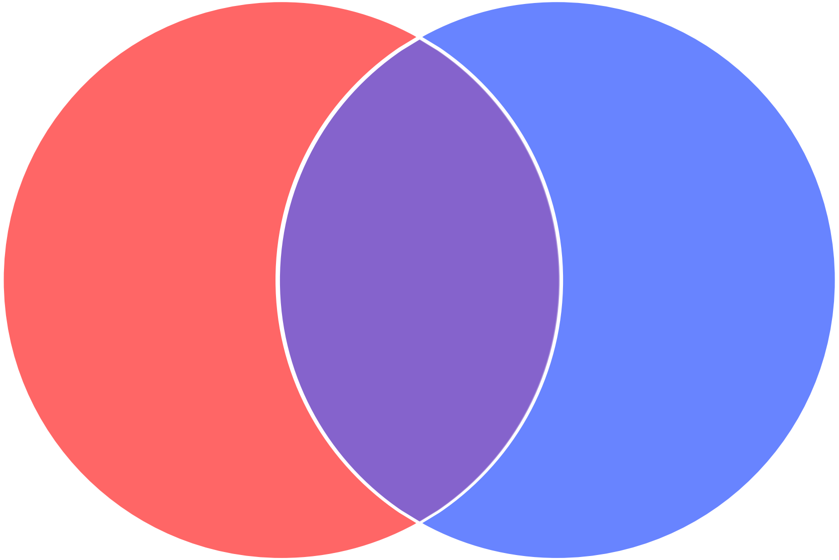
BAYES THEOREM DERIVATION



By conditional and joint relationship:

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

BAYES THEOREM DERIVATION



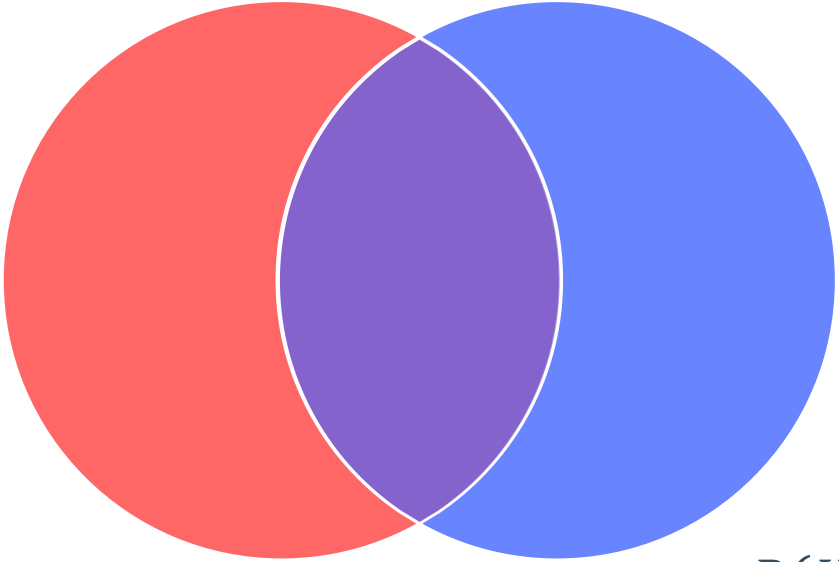
By conditional and joint relationship:

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

To invert conditional probability:

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

BAYES THEOREM DERIVATION



By conditional and joint relationship:

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

To invert conditional probability:

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

$$P(X) = \sum_{Z} P(X,Z) = \sum_{Z} P(X|Z) * P(Z)$$

BAYES THEOREM

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

BAYES THEOREM

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

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

NAÏVE BAYES CLASSIFICATION

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

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

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

$$P(C|X) = P(X_1, X_2, \ldots, X_n|C) * P(C)$$

$$P(X_1|X_2, \ldots, X_n, C) * P(X_2, \ldots, X_n|C) * P(C)$$

$$\ldots$$

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)

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

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

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

TRAINING NAÏVE BAYES

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

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

Means select potential class with largest value

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

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

THE LOG TRICK

Multiplying many values together causes computational instability (underflows)

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

THE LOG TRICK

Multiplying many values together causes computational instability (underflows)

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

Work with log values and sum the results

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

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(Play=Yes) = 9/14

P(Play=No) = 5/14

EXAMPLE: TRAINING NAÏVE BAYES TENNIS MODEL

P(Play=Yes) = 9/14

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

P(Play=No) = 5/14

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

EXAMPLE: TRAINING NAÏVE BAYES TENNIS MODEL

P(Play=Yes) = 9/14

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

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

P(Play=No) = 5/14

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

Wind	Play=Yes	Play=No
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'=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

$$P(yes|sunny, cool, high, strong) = P(sunny|yes) * P(cool|yes) * P(high|yes) * P(strong|yes) * P(yes)$$

$$P(no|sunny, cool, high, strong) = P(sunny|no) * P(cool|no) * P(high|no) * P(strong|no) * P(no)$$

EXAMPLE: PREDICTING TENNIS WITH NAÏVE BAYES

Predict outcome for the following:

x'=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

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

EXAMPLE: PREDICTING TENNIS WITH NAÏVE BAYES

Predict outcome for the following:

x'=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=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
Overall Label	**9/14**	**5/14**

EXAMPLE: PREDICTING TENNIS WITH NAÏVE BAYES

Predict outcome for the following:

x'=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=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
Overall Label	**9/14**	**5/14**
Probability	**0.0053**	**0.0206**

EXAMPLE: PREDICTING TENNIS WITH NAÏVE BAYES

Predict outcome for the following:

x'=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=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
Overall Label	**9/14**	**5/14**
Probability	**0.0053**	**0.0206**

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

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

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

0

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

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

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

TYPES OF NAÏVE BAYES

Naïve Bayes Model	Data Type
Bernoulli	Binary (T/F)

TYPES OF NAÏVE BAYES

Naïve Bayes Model	Data Type
Bernoulli	Binary (T/F)
Multinomial	Discrete (e.g. count)

TYPES OF NAÏVE BAYES

Naïve Bayes Model	Data Type
Bernoulli	Binary (T/F)
Multinomial	Discrete (e.g. count)
Gaussian	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)
Solution	▪ **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)
Solution	▪ **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
```

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)
```

Laplace smoothing parameter

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)
```

Laplace smoothing parameter

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

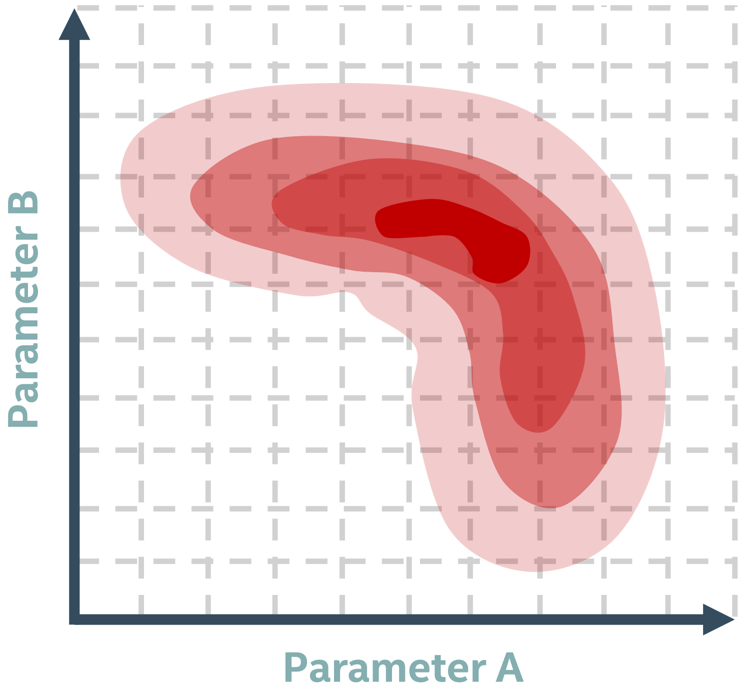
```
BNB = BNB.fit(X_train, y_train)
y_predict = BNB.predict(X_test)
```

Other naïve Bayes models: `MultinomialNB`, `GaussianNB`.

GRID SEARCH AND PIPELINES

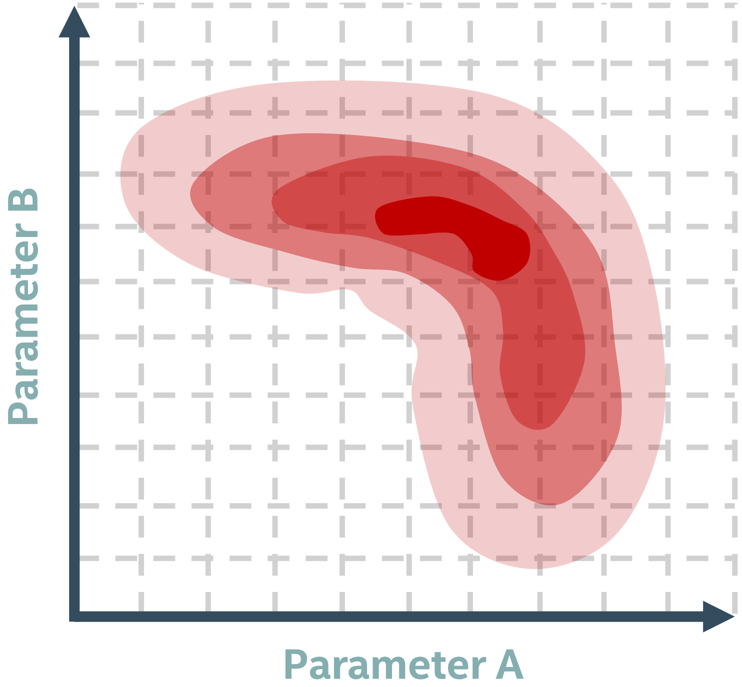
GENERALIZED HYPERPARAMETER GRID SEARCH

- 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
```

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
```

logistic regression method

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)
```

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)?

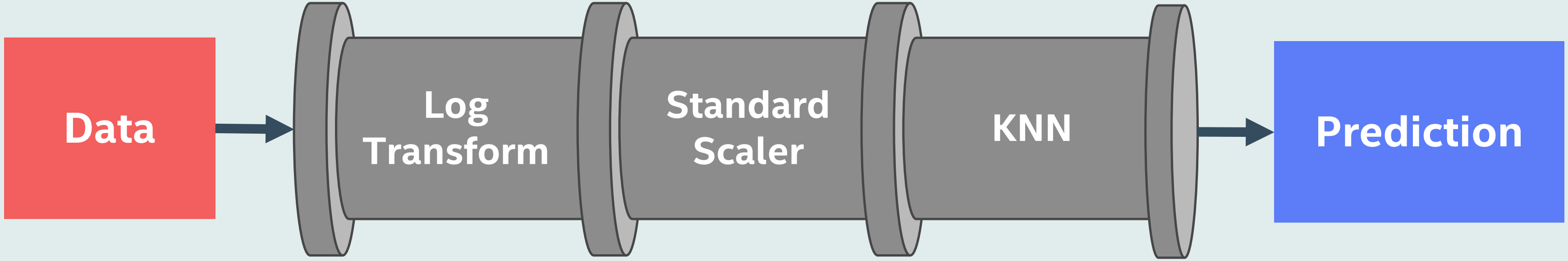
OPTIMIZING THE REST OF THE PIPELINE

- 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!

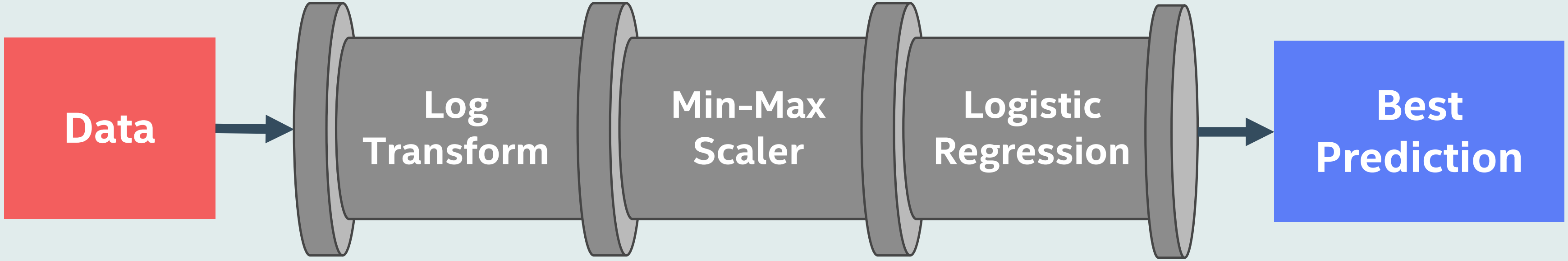
AUTOMATING MACHINE LEARNING WITH PIPELINES

- Machine learning models often selected empirically



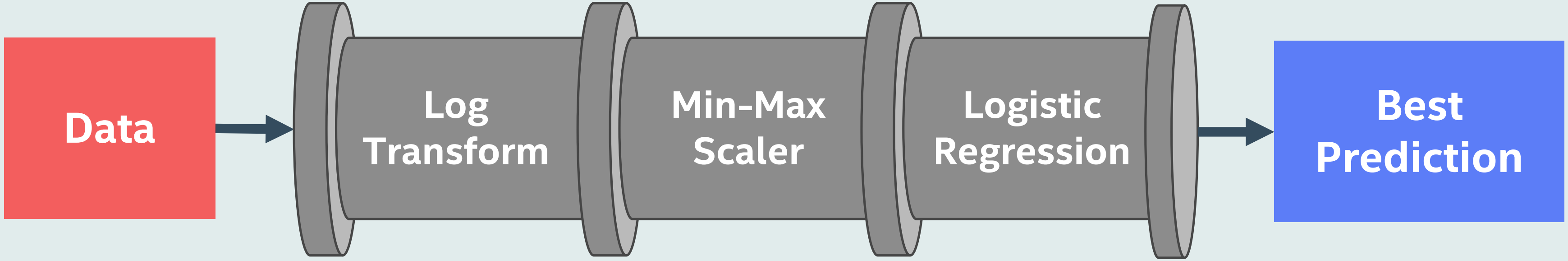
AUTOMATING MACHINE LEARNING WITH PIPELINES

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



AUTOMATING MACHINE LEARNING WITH PIPELINES

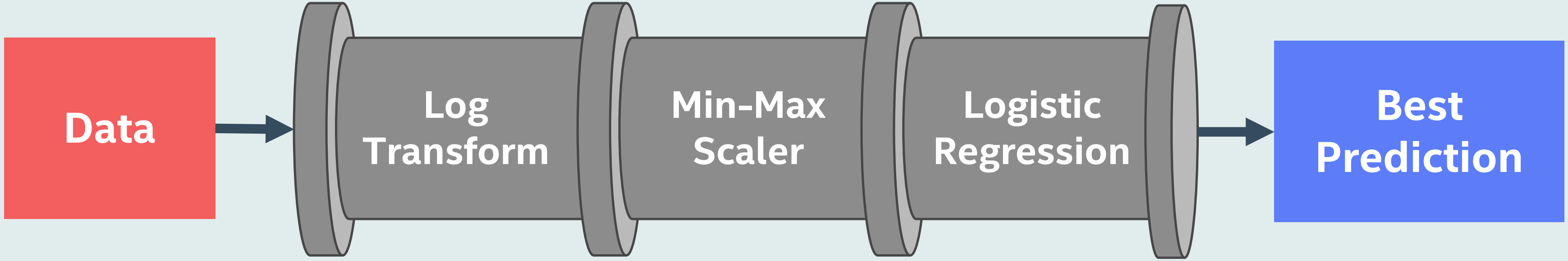
- Machine learning models often selected empirically
- By trying different processing methods and tuning multiple models



How to automate this process?

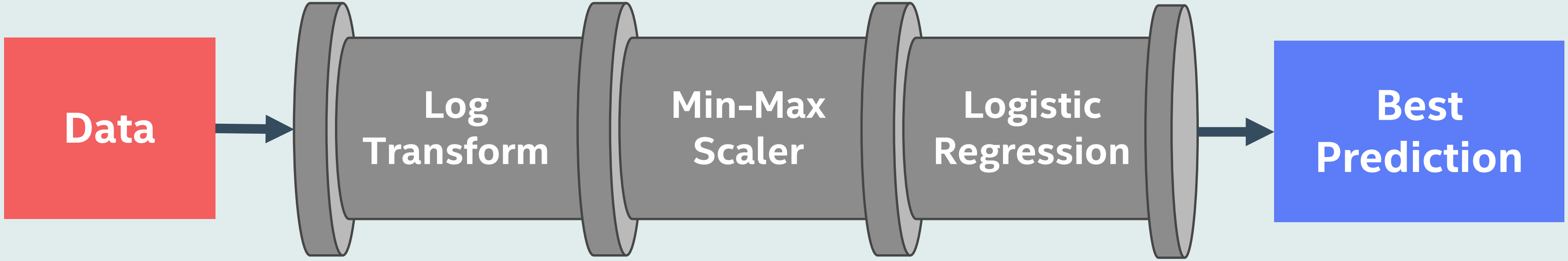
AUTOMATING MACHINE LEARNING WITH PIPELINES

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



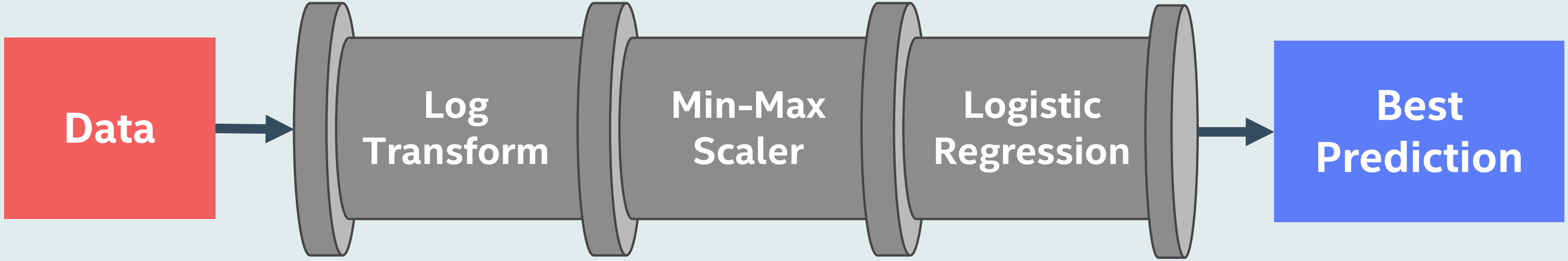
AUTOMATING MACHINE LEARNING WITH PIPELINES

- 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
```

feature
scaler class

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
```

lasso
model
class

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` intel® Nervana™ AI Academy