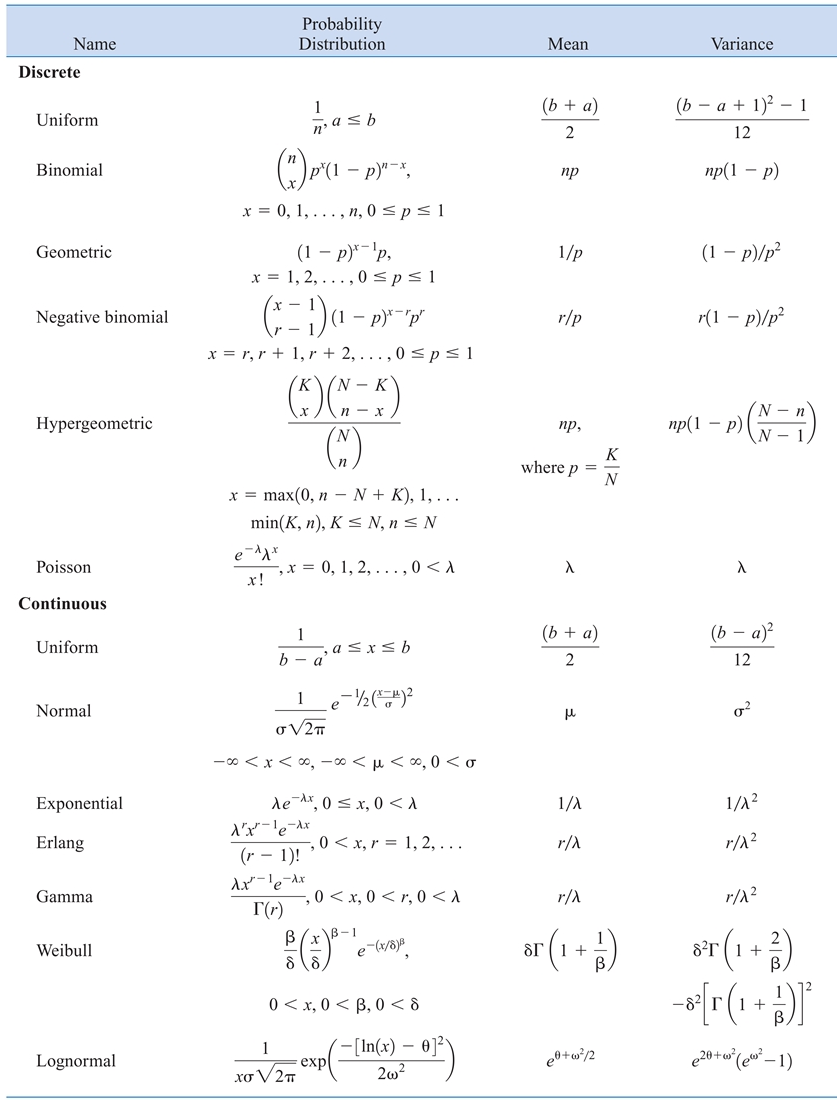
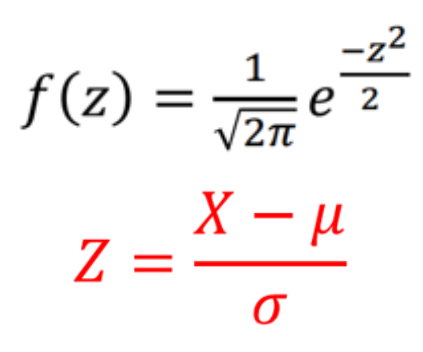
Basic formulas for statistical measures

Summary of Common Probability Distributions



|  |  |  |  |
| --- | --- | --- | --- |
| Law | Discrete | Continuous | |
| Mean (μ)  Expected Value | μ = E(X) = Σ (x f(x)) | μ = E(X) = ∫ (x f(x)) dx | |
| Joint Mean Expected Value | μg(x,y) = ΣΣ (g(x,y) \* f(x,y)) | μg(x,y) = ∫∫ (g(x,y) \* f(x,y)) dx dy | |
| Variance (σ^2) | σ2 = Σ (x2 \* f(x)) - μ2 | σ2 = ∫ (x2 \* f(x)) dx - μ2 | |
| σ2 = E (x2) - μ2 | | |
| Covariance (σxy)  check linearity  -∞ <σxy <∞ | σxy = ΣΣ [(xy) f(x,y)] - μxμy | σxy = ∫∫ [(xy) f(x,y)] dx dy - μx μy | |
| μx = Σ ( x \* g(x) )  μy = Σ ( y \* h(y) ) | μx = ∫ ( x \* g(x) ) dx  μy = ∫ ( y \* h(y) ) dy | |
| σxy = E(XY) - μx μy | | |
| Correlation (ρxy)  Measure linearity  -1 < ρxy <1 | σx2 = E(X2) - μx2 | | σy2 = E(y2) – μy2 |
| ρxy = σxy / (σx σy) | | |

* Binominal (Bernolli) used for x success in n trials (true and false trials)
* Geometric is used for x trials to get the first success
* Negative binonminal is used for kth success in x trials
* Poission is used for number of outcomes in a region with average or rate
* Uniform is used when the outcomes have equal likelihood
* Exponential is used for time between events with average
* Standard Normal Distribution has the following equation to convert its table0 values to any other normal distribution system



* Where μ is the mean of population
* σ is the standard deviation of the population
* x is the value from foreign system with normal distribution
* z is the result equivalent from standard normal distribution

Frequency Table

• Range = Upper value – Lower value.

• Sturges's formula for class count: k =1+3.322log(n) 🡪 n number of items.

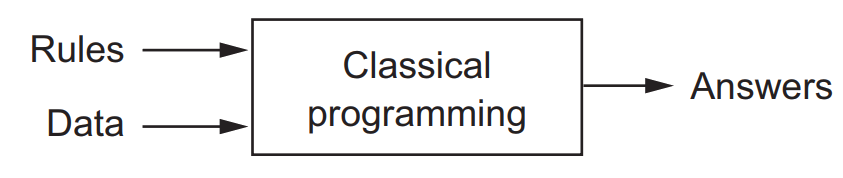
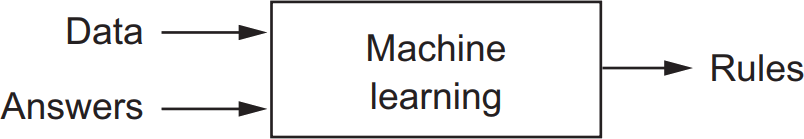
• Calculate class width: R / K.

• Assign data points to classes to build the frequency table.

Quartile

* Count the data points.
* Order them from smallest to largest.
* Locate the median (Q2).
* Divide the dataset into two at the median.
* For odd-sized datasets, exclude the median for the next steps.
* For even-sized datasets, include both middle values in the halves.
* The median of the lower half is Q1.
* The median of the upper half is Q3

|  |  |  |  |
| --- | --- | --- | --- |
|  | Category | Algorithm | When to Use |
|  | **Sorting** | Bubble Sort | Simple sorting, small datasets |
| Insertion Sort | Small datasets, mostly sorted data |
| Selection Sort | Small datasets, simplicity over efficiency |
| Merge Sort | Large datasets, require stable sort |
| Quick Sort | Large datasets, average case speed |
| Heap Sort | Need to sort large datasets in place |
| Radix Sort | Large datasets with fixed-length keys |
| Bucket Sort | Large datasets, uniformly distributed |
| External Sort | Very large datasets that don’t fit in memory |
|  | **Searching** | Linear Search | Unsorted data, small datasets |
| Binary Search | Sorted data, large datasets |
| Depth-First Search (DFS) | Graphs, exploring edges/node paths |
| Breadth-First Search (BFS) | Graphs, finding shortest path |
|  | **Data Structures** | Arrays, Linked Lists, queue, stack, priority queue | |
|  | **String Manipulation** |  | |
| Hierarchical data | **Tree and BST** | Tree Traversal | Accessing/Searching/Manipulating tree data |
| BST Operations | Efficient search, insert, delete in sorted data |
| AVL Trees | Self-balancing BST where lookup speed is critical |
| networks, paths, and relationships | **Graph** | Dijkstra's Algorithm | Shortest path in weighted graph without negative edges |
| Kruskal's Algorithm | Minimum spanning tree in sparse graphs |
| Prim's Algorithm | Minimum spanning tree in dense graphs |
| sequence of decisions | **Greedy** |  | |
| Can be divided to sub problems | **Divide and conquer** | Quick Sort | When average case performance is key |
| Merge Sort | Large datasets where stability is key |
| Binary Search | Quickly finding an item in sorted data |
| Strassen's Matrix Multiplication | Large matrix multiplication |
| Karatsuba Algorithm | Fast multiplication of large numbers |
| Permutations Combinations | **Backtracking** |  |  |
| Overlapping Subproblems Optimal Substructure | **Dynamic Programming** | Fibonacci, knapsack and palindromes | |

`

1. **Type**:
   * **Supervised Learning**: Algorithms learn from labeled data, with input-output pairs provided during training.
   * **Unsupervised Learning**: Algorithms learn patterns and structures in unlabeled data without explicit supervision.
   * **Reinforcement Learning**: Agents learn to interact with an environment to maximize rewards through trial and error.
2. **Functionality**:
   * **Classification**: Algorithms predict discrete class labels for given input data.
   * **Regression**: Algorithms predict continuous values based on input data.
   * **Clustering**: Algorithms group similar data points together based on some similarity measure.
   * **Dimensionality Reduction**: Techniques reduce the number of input variables while preserving important information.
3. **Model Complexity**:
   * **Parametric Models**: Models have a fixed number of parameters and a predefined structure, such as linear regression.
   * **Non-parametric Models**: Models have a flexible number of parameters, such as decision trees and k-nearest neighbors.
   * **Deep Learning**: Neural networks with multiple hidden layers, capable of learning intricate patterns and representations from data.
4. **Domain or Application**:
   * **Computer Vision**: Algorithms designed to process and interpret visual data.
   * **Natural Language Processing (NLP)**: Algorithms designed to understand and process human language.
   * **Speech Recognition**: Algorithms designed to transcribe spoken language into text.
   * **Recommendation Systems**: Algorithms designed to recommend items or content to users based on their preferences.

CNN and RNNS for high-dimensional, large, complex dataset (low interpretability and higher computational resources)

* Let's say you're working on predicting the price of a house based on various features such as square footage, number of bedrooms, number of bathrooms, and location. In this case, you have a continuous target variable (price) that you want to predict, and the relationship between the features and the target variable is likely to be linear.
* Using linear regression would be appropriate in this scenario because it models the relationship between the input features and the target variable as a linear function. Linear regression would allow you to estimate the coefficients of each feature, which represent the change in the target variable for a one-unit change in each feature, holding all other features constant.
* On the other hand, logistic regression is used for binary classification problems where the target variable is categorical (e.g., yes/no, 1/0). If the target variable is not binary or if the relationship between the features and the target variable is not linear, logistic regression would not be suitable.
* Let's consider a scenario where you have a dataset with high-dimensional features, such as images represented as pixel values. You want to perform binary classification to distinguish between images of cats and dogs. Each image is represented by a large number of features (e.g., pixel values), making the dataset high-dimensional.
* In this case, using SVM with a kernel function, such as the radial basis function (RBF) kernel, can effectively handle high-dimensional data and find complex nonlinear decision boundaries between the classes. SVM with the RBF kernel is capable of capturing intricate patterns in the data, which may be necessary for distinguishing between images of cats and dogs with high accuracy.
* On the other hand, PCA is a dimensionality reduction technique that aims to reduce the dimensionality of the dataset by finding a lower-dimensional representation while preserving most of the variance in the data. However, PCA may not be as effective in capturing complex nonlinear relationships in the data, especially in cases where the decision boundary between classes is highly nonlinear or intricate.
* Consider a text classification problem where you want to classify emails as either spam or not spam based on the words contained in the email. Each email is represented by a bag-of-words representation, where the presence or absence of words in the email is used as features for classification. In this scenario:
* Naive Bayes: Naive Bayes classifiers are well-suited for text classification tasks because they assume that features are conditionally independent given the class label. Despite this simplifying assumption, Naive Bayes classifiers often perform well on text data and are computationally efficient. They calculate the probability of each class given the features and select the class with the highest probability.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algorithm | Type | Functionality | Deep Learning | Model Complexity | Code example | image |
| CNN | Supervised | Classification, Regression | Yes | Deep | from keras.models import Sequential  from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense  model = Sequential()  model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(MaxPooling2D((2, 2)))  model.add(Conv2D(64, (3, 3), activation='relu')) model.add(MaxPooling2D((2, 2)))  model.add(Flatten())  model.add(Dense(64, activation='relu'))  model.add(Dense(10, activation='softmax')) | Understanding Convolutional Neural Network: A Complete Guide |
| SVM (Support Vector Machine) | Supervised | Classification, Regression | No | Parametric | from sklearn import datasets  from sklearn.model\_selection  import train\_test\_split from sklearn.svm  import SVC  iris = datasets.load\_iris()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=42)  svm\_model = SVC(kernel='linear')  svm\_model.fit(X\_train, y\_train)  accuracy = svm\_model.score(X\_test, y\_test) print("Accuracy:", accuracy) | Support vector machine - Wikipedia |
| Decision Forests | Supervised | Classification, Regression | No | Non-parametric | from sklearn import datasets  from sklearn.model\_selection  import train\_test\_split  from sklearn.ensemble import RandomForestClassifier  iris = datasets.load\_iris()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=42)  rf\_model = RandomForestClassifier(n\_estimators=100) rf\_model.fit(X\_train, y\_train)  accuracy = rf\_model.score(X\_test, y\_test)  print("Accuracy:", accuracy) | Decision forests in TensorFlow. Machine learning extends beyond Deep… | by  Ibrahim Olagoke | Medium |
| Naive Bayes | Supervised | Classification | No | Parametric | from sklearn import datasets  from sklearn.model\_selection  import train\_test\_split  from sklearn.naive\_bayes  import GaussianNB  iris = datasets.load\_iris()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=42)  nb\_model = GaussianNB()  nb\_model.fit(X\_train, y\_train)  accuracy = nb\_model.score(X\_test, y\_test) print("Accuracy:", accuracy) | Naïve Bayes Algorithm. Exploring Naive Bayes: Mathematics, How… | by  Bassant Gamal | Analytics Vidhya | Medium |
| Linear Regression | Supervised | Regression | No | Parametric | from sklearn.datasets import load\_boston  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  boston = load\_boston()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(boston.data, boston.target, test\_size=0.3, random\_state=42)  lr\_model = LinearRegression()  lr\_model.fit(X\_train, y\_train)  accuracy = lr\_model.score(X\_test, y\_test)  print("Accuracy:", accuracy) |  |
| Logistic Regression | Supervised | Classification | No | Parametric | from sklearn import datasets  from sklearn.model\_selection  import train\_test\_split  from sklearn.linear\_model  import LogisticRegression  iris = datasets.load\_iris()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=42)  lr\_model = LogisticRegression()  lr\_model.fit(X\_train, y\_train)  accuracy = lr\_model.score(X\_test, y\_test)  print("Accuracy:", accuracy) |  |
| KNN (K-Nearest Neighbors) | Supervised | Classification | No | Non-parametric | from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split  from sklearn.neighbors import KNeighborsClassifier  iris = load\_iris()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=42)  knn\_model = KNeighborsClassifier(n\_neighbors=5)  knn\_model.fit(X\_train, y\_train)  accuracy = knn\_model.score(X\_test, y\_test)  print("Accuracy:", accuracy) | K-Nearest Neighbor (KNN) Explained | Machine Learning Archive |
| PCA (Principal Component Analysis) | Unsupervised | Dimension Reduction | No | Non-parametric | from sklearn.datasets import load\_iris  from sklearn.decomposition import PCA  iris = load\_iris()  pca = PCA(n\_components=2)  X\_reduced = pca.fit\_transform(iris.data) | Principal Component Analysis (PCA) 101 - NumXL |
| K-Means | Unsupervised | Clustering | No | Non-parametric | from sklearn.datasets import make\_blobs  from sklearn.cluster import KMeans  X, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=0)  kmeans = KMeans(n\_clusters=4)  kmeans.fit(X) | 2 K-means clustering | Machine Learning for Biostatistics |
| GANs (Generative Adversarial Networks) | Unsupervised | Generation | Yes | Deep |  | Generative Adversarial Networks and Some of GAN Applications: Everything  You Need to Know |
| Autoencoder | Unsupervised | Dimension Reduction, Generation | Yes | Deep |  | IoT | Free Full-Text | Deep Autoencoder-Based Integrated Model for Anomaly  Detection and Efficient Feature Extraction in IoT Networks |

**Advanced CNNs**

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Type | Best Case to Use | Model Size |
| SqueezeNet | CNN | When limited computational resources are available | Small |
| MobileNet | CNN | Mobile and edge devices, real-time applications | Small |
| BiLSTM (Bidirectional LSTM) | RNN | Sequential data with bidirectional context | Medium |
| GRU (Gated Recurrent Unit) | RNN | Sequences with less complex dependencies | Medium |
| LSTM (Long Short-Term Memory) | RNN | Long sequences with complex dependencies | Medium |
| DenseNet | CNN | Image classification tasks | Large |
| EfficientNet | CNN | State-of-the-art performance with efficient use of resources | Large |
| NASNet (Neural Architecture Search Network) | CNN | High-performance image classification tasks | Large |
| AlexNet | CNN | Legacy image classification tasks | Large |
| GoogLeNet (Inception) | CNN | Tasks requiring strong generalization | Large |
| VGG (Visual Geometry Group) networks (e.g., VGG16, VGG19) | CNN | Image classification tasks, benchmarking | Large |
| Attention-based models (e.g., Transformer with Attention Mechanisms) | RNN/CNN | NLP tasks, sequence-to-sequence learning | Varies |
| Transformer | RNN/CNN | NLP tasks, language translation | Varies |

**Simple list:**

1. SVM (Support Vector Machine)
2. Decision Forests
3. Naive Bayes
4. Linear Regression
5. Logistic Regression
6. KNN (K-Nearest Neighbors)
7. PCA (Principal Component Analysis)
8. GANs (Generative Adversarial Networks)
9. Autoencoder
10. K-Means
11. AlexNet
12. VGG (Visual Geometry Group) networks (e.g., VGG16, VGG19)
13. GoogLeNet (Inception)
14. ResNet (Residual Network)
15. DenseNet
16. MobileNet
17. EfficientNet
18. NASNet (Neural Architecture Search Network)
19. SqueezeNet
20. LSTM (Long Short-Term Memory)
21. GRU (Gated Recurrent Unit)
22. BiLSTM (Bidirectional LSTM)
23. Transformer
24. Attention-based models (e.g., Transformer with Attention Mechanisms)