MACHINE LEARNING

Augustin Cablant

August 2023

Table des matières

1 Introduction

1.1 What is Machine Learning ?

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

In general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data.

An error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model. If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this "evaluate and optimize" process, updating weights autonomously until a threshold of accuracy has been met.

Supervised learning, also known as supervised machine learning, is defined by its use of labeled datasets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, the model adjusts its weights until it has been fitted appropriately. This occurs as part of the cross validation process to ensure that the model avoids overfitting or underfitting. Supervised learning helps organizations solve a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox. Some methods used in supervised learning include neural networks, naïve bayes, linear regression, logistic regression, random forest, and support vector machine (SVM).

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. This method's ability to discover similarities and differences in information make it ideal for exploratory data analysis, cross-selling strategies, customer segmentation, and image and pattern recognition. It's also used to reduce the number of features in a model through the process of dimensionality reduction. Principal component analysis (PCA) and singular value decomposition (SVD) are two common approaches for this. Other algorithms used in unsupervised learning include neural networks, k-means clustering, and probabilistic clustering methods.

Semi-supervised learning offers a happy medium between supervised and unsupervised learning. During training, it uses a smaller labeled data set to guide classification and feature extraction from a larger, unlabeled data set. Semi-supervised learning can solve the problem of not having enough labeled data for a supervised learning algorithm. It also helps if it's too costly to label enough data.

Reinforcement machine learning is a machine learning model that is similar to supervised learning, but the algorithm isn't trained using sample data. This model learns as it goes by using trial and error. A sequence of successful outcomes will be reinforced to develop the best recommendation or policy for a given problem.

1.2 Training a perceptron

First steps :
- Selecting features and collecting data labeled training examples
- Choosing a performance metric
- Choosing a classifier and optimization algorithm
- Evaluating the performance of the model
- Tuning the algorithm

Code tricks :
- Use np.unique(y) to return the unique class labels
- Use from sklearn.model-selection import train-test-split
- Use np.bincount(y) to count the number of occurrences of each value. We want to check the proportions of class labels in the training and the test set. Then, if it's the case, we can use in our train-test-split the built-in support for stratification via stratify = y
- For optimal performance in the **gradient descent** we want to scale our features :

```
from sklearn.model_selection import StandardScaler()
sc = StandardScaler()
sc.fit(X_train)
X_train_std, X_test_std = sc.transform(X_train), sc.transform(X_test)
```

2 Linear Regression

2.1 What is Linear Regression ?

Linear regression is a statistical method used to examine the relationship between two continuous variables : one independent variable and one dependent variable. The goal of linear regression is to find the best-fitting line through a set of data points, which can then be used to make predictions about future observations.

The equation for a simple linear regression model is : $y = b_0 + b_1.x$.where y is the dependent variable, x is the independent variable, b0 is the y-intercept (the point at which the line crosses the y-axis), and b1 is the slope of the line. The slope represents the change in y for a given change in x.

To determine the best-fitting line, we use the method of least squares, which finds the line that minimizes the sum of the squared differences between the predicted y values and the actual y values.

Linear regression can also be extended to multiple independent variables, known as multiple linear regression. The equation for a multiple linear regression model is : $y = b_0 + b_1x_1 + b_2x_2 + \ldots + b_nx_n$ where $x_1, x_2, \ldots, x_n$ are the independent variables, and $b_1, b_2, \ldots, b_n$ are the corresponding coefficients.

Linear regression can be used for both simple linear regression and multiple linear regression problems. The coefficients b_0 and $b_1, \ldots, b_n$ are estimated using the method of least squares. Once the coefficients are estimated, they can be used to make predictions about the dependent variable.

Linear regression can be used to make predictions about the future, such as predicting the price of a stock or the number of units of a product that will be sold. However, linear regression is a relatively simple method and may not be appropriate for all problems. It assumes that the relationship between the independent and dependent variables is linear, which may not always be the case.

Additionally, Linear Regression is highly sensitive to outliers, meaning if there are any extreme values that don't follow the general trend of the data it will significantly impact the accuracy of the model.

In conclusion, linear regression is a powerful and widely used statistical method that can be used to examine the relationship between two continuous variables. It is a simple, yet powerful tool that can be used to make predictions about the future. However, it is important to keep in mind that linear regression assumes a linear relationship between the variables and is sensitive to outliers, which can impact the accuracy of the model.

2.2 What are the assumptions of linear regression ?

The assumptions of linear regression are :

Linearity : The relationship between the independent and dependent variables is linear.

Independence : The observations are independent of each other.

Homoscedasticity : The variance of the error term is constant across all levels of the independent variables.

Normality : The error term is normally distributed.

No multicollinearity : The independent variables are not highly correlated with each other.

No autocorrelation : The error term is not autocorrelated with itself.

2.3 How do you determine the goodness of fit of a linear regression model ?

There are several ways to determine the goodness of fit of a linear regression model :

R-squared : R-squared is a statistical measure that represents the proportion of the variance in the dependent variable that is explained by the independent variables in the model. An R-squared value of 1 indicates that the model explains all the variance in the dependent variable, and a value of 0 indicates that the model explains none of the variances.
A data set has n values marked $y_1, ..., y_n$ (collectively known as y_i or as a vector $y = [y_1, ..., y_n].T$), each associated with a fitted (or modeled, or predicted) value $f_1, ..., f_n$ (known as f_i, or sometimes $\hat{y}_i$, as a vector f).

Let's define :

- The **residuals** as $e_i = y_i - f_i$ (forming a vector e)

- The **residual sum of squares** as $SS_{res} = \sum_i e_i^2$

- The **total sum of squares** as $SS_{res} = \sum_i (y_i - \overline{y})^2$

Then, the **coefficient of determination** as $R^2 = 1 - \frac{SS_{res}}{SS_{res}}$. In the best case, the modeled values exactly match the observed values, which results in $SS_{\text{res}} = 0$ and $R^2 = 1$.
In a general form, R^2 can be seen to be related to the fraction of variance unexplained (FVU), since the second term compares the unexplained variance (variance of the model's errors) with the total variance (of the data).
R^2 is a measure of the goodness of fit of a model. In regression, the R^2 coefficient of determination is a statistical measure of how well the regression predictions approximate the real data points. An R^2 of 1 indicates that the regression predictions perfectly fit the data.

Adjusted R-squared : Adjusted R-squared is a modified version of R-squared that accounts for the number of independent variables in the model. It is a better indicator of the model's goodness of fit when comparing models with different numbers of independent variables.
The use of an adjusted R2 (one common notation is $\overline{R^2}$) is an attempt to account for the phenomenon of the R^2 automatically increasing when extra explanatory variables are added to the model.
$\overline{R^2} = 1 - \frac{SS_{res}/d_{res}}{SS_{tot}/d_{tot}}$ where where $dfres$ is the degrees of freedom of the estimate of the population variance around the model, and $dftot$ is the degrees of freedom of the estimate of the population variance around the mean.

Root Mean Squared Error (RMSE) : RMSE measures the difference between the predicted values and the actual values. A lower RMSE indicates a better fit of the model to the data.
$\text{RMSD} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$.

Mean Absolute Error (MAE) : MAE measures the average difference between the predicted values and the actual values. A lower MAE indicates a better fit of the model to the data.
$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$.

2.4 How do you deal with outliers in linear regression ?

Outliers in linear regression can have a significant impact on the model's predictions, as they can skew the regression line. There are several ways to deal with outliers in linear regression, including :

A - Removing outliers :
One option is to simply remove outliers from the dataset before training the model. Howe-

ver, this can lead to the loss of important information.

- **max trials** refers to the maximum number of iterations
- **min samples** set the minimum number of the randomly chosen training
- **loss parameter** is 'absolute loss', it means that the algorithm computes absolute vertical distances between the fitted line and the training examples
- **the residual threshold** allows the training examples to be included in the inlier set if their vertical distance to the fitted line is within **residual threshold** distance units
- By default, scikit-learn uses the median absolute deviation (MAD) estimate to select the inlier threshold.

B - Transforming the data :
Applying a transformation such as taking the log of the data can help to reduce the impact of outliers.

C - Using robust regression methods :
Robust regression methods, such as RANSAC or Theil-Sen, are less sensitive to outliers than traditional linear regression.
The **RANdom SAmple Consensus (RANSAC)** algorithm fits a regression model to a subset of the data. To sum up, the RANSAC algorithm :
- Select a random number of examples to be inliers and fit the model
- Test all other data points against the fitted model and add those points that fall within a user-given tolerance to the inliers
- Refit the model using all inliers
- Estimate the error of the fitted model versus the inliers
Terminate the algorithm if the performance meets a certain user-defined threshold or if a fixed number of iterations were reached ; go back to step 1 otherwise.

```
from sklearn.linear_model import RANSACRegressor

ransac = RANSACRegressor(LinearRegression(),
                         max_trials = 100,
                         min_samples = 50,
                         loss = 'absolute_loss',
                         residual_threshold = 5.0,
                         random_state = 0)
ransac.fit(X,y)
```

D - Using regularization : Regularization can help to prevent overfitting, which can be caused by outliers, by adding a penalty term to the cost function.

The most popular approaches to regularized linear regression are the so-called **Ridge Regression, least absolute shrinkage and selection operator (LASSO), and elastic Net**.

Ridge Regression is an L2 penalized model where we simply add the squared sum of the weights to our least-squares cost function : $J(w)_{ridge} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \alpha||w||_2^2$.
By increasing the value of hyperparameter α we increase the regularization strength and thereby shrink the weights of our model (note that we don't regularize the intercept term, w_0).

LASSO is an L1 penalized model : $J(w)_{LASSO} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \alpha||w||_1$.

Elastic net is a good compromise : $J(w)_{ElasticNet} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \alpha_1||w||_2^2 + +\alpha_2||w||_1$.

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)

from sklearn.linear_model import Lasso
lasso = Lasso(alpha=1.0)

from sklearn.linear_model import ElasticNet
elanet = ElasticNet(alpha=1.0, l1_ratio = 0.5)
```

2.5 How do you deal with non-linear relationship between explanatory and response variable ?

One way is to use **polynomial regression** model by adding polynomial terms :
$y = w_0 + w_1x^1 + w_2x^2 + ... + w_dx^d$.

```
# Given X,y
quadratic = PolynomialFeatures(degree=2)
cubic = (degree=3)
X_quad = quadratic.fit_transform(X)
X_cubic = cubic.fit_transform(X)
```

An other way is to use log-transformation :

```
# Given X,y
X_log = np.log(X)
y_log = np.log(y)
```

3 Logistic Regression

Logistic Regression is a statistical method used for predicting binary outcomes, such as success or failure, based on one or more independent variables. It is a popular technique in machine learning and is often used for classification tasks, such as determining whether an email is spam or not, or predicting whether a customer will churn.

The logistic regression model is based on the logistic function, which is a sigmoid function that maps the input variables to a probability between 0 and 1. The probability is then used to make a prediction about the outcome.

The logistic regression model is represented by the following equation :
$P(y = 1 \mid x) = \frac{1}{1 + e^{-(b_0 + b_1.x_1 ... + b_n.x_n)}}$ where $P(y = 1 \mid x)$ is the probability that the outcome y is 1 given the input variables x, b_0 is the intercept, and $b_1, b_2, \ldots, b_n$ are the coefficients for the input variables $x_1, x_2, \ldots, x_n$.

The coefficients are determined by training the model on a dataset and using a optimization algorithm, such as gradient descent, to minimize the cost function, which is typically the log loss.

Once the model is trained, it can be used to make predictions by inputting new data and calculating the probability of the outcome being 1. The threshold for classifying the outcome as 1 or 0 is typically set at 0.5, but this can be adjusted depending on the specific task and desired trade-off between false positives and false negatives.

3.1 Question 1 - What is the logistic function ?

The logistic function, also known as the **sigmoid function**, is an S-shaped curve that maps any real-valued number to a value between 0 and 1. It is defined as $f(x) = \frac{1}{(1+e^{-x})}$. The logistic function is used in logistic regression to model the probability of a **binary outcome**.

3.2 Question 2 - Can logistic regression be used for multiclass classification ?

Yes, logistic regression can be used for multiclass classification by creating a separate binary logistic regression model for each class and choosing the class with the highest predicted probability. This is known as *one-vs-all* or *one-vs-rest approach*. Alternatively, we can use **softmax regression** which is a generalization of logistic regression which can handle multiple classes directly.

3.3 Question 3 - How do you interpret the coefficients in logistic regression ?

The coefficients in logistic regression represent the change in the log odds of the outcome for a one-unit change in the predictor variable while holding all other predictors constant. The odds ratio can be used to interpret the magnitude of the coefficients. An odds ratio greater than 1 indicates that a unit increase in the predictor increases the odds of the outcome, while an odds ratio less than 1 indicates that a unit increase in the predictor decreases the odds of the outcome.

3.4 Tackling overfitting via regularization

Overfitting is a common problem in ML, where a model performs well on training data but does not generalize well to unseen data. If a model suffers from **overfitting**, we also say that the model has a high variance, which can be caused by having too many parameters, leading to a model that is too complex, given the underlying data. Similarly, our model can also suffer from **underfitting** (high bias), which means that our model is not complex enough to caputre the pattern in the training data well and therefore also suffers from low performance on unseen data.
Often, researchers use the terms "bias" and "variance" or "bias-variance tradeoff" to describe the performance of a model. In general, we might say that "high variance" is proportional to overfitting and "high bias" is proportional to underfitting.

In the context of ML models, **variance** measures the consistency (or variability) of the model prediction for classifying a particular example if we retrain the model multiple times, for example, on different subsets of the training dataset. We can say that the model is sensitive to the randomness in the training data. In contrast, **bias** measures how far off the predictions are from the correct values in general if we rebuild the model multiple times on different training datasets ; bias is the measure of the systematic error that is not due to randomness.

On way of finding a good bias-variance tradeoff is to tune the complexity of the model via regularization. Regularization is a very useful method for handling collinearity (high correlation among features), filtering out noise from data, and eventually preventing overfitting. The most common form of regularization is so-called **L2 regularization** : $\frac{\lambda}{2}||w||_2^2$ where λ is the **regularization-parameter**.
Note that regularization is another reason why feature scaling such as standardization is important. For regularization to work properly, we need to ensure that all our features are on comparable scales.
The cost function for logistic regression can be regularized by adding this simple regularization term, which will shrink the weights during model training :

$$J(w) = \sum_{i=1}^{n}(-y^{(i)}log(\phi(z^{(i)}) - (1 - y^{(i)})log(1 - \phi(z^{(i)}))) + \frac{\lambda}{2}||w||_2^2.$$

The parameter C, that is implemented for the LogisticRegression() in scikit-learn comes from a convention in support vector machines. This term is directly related to the regularization parameter λ, which is its inverse. Consequently, decreasing the values of the inverse regularization parameter, C, means that we are increasing the regularization strength.

4 Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression problems. The main idea behind SVMs is to find the boundary that separates different classes in the data by maximizing the margin, which is the distance between the boundary and the closest data points from each class. These closest data points are called support vectors.
The optimization objective is to maximize the margin (defined as the distance between the separating hyperplane or "decision boundary" and the training examples that are closest to this hyperplane, which are the so-called **support vectors**.
SVMs are particularly useful when the data is not linearly separable, which means that it cannot be separated by a straight line. In these cases, SVMs can transform the data into a higher dimensional space using a technique called kernel trick, where a non-linear boundary can be found. Some common kernel functions used in SVMs are polynomial, radial basis function (RBF), and sigmoid.

One of the main advantages of SVMs is that they are very effective in high-dimensional spaces and have a good performance even when the number of features is greater than the number of samples. Additionally, SVMs are memory-efficient because they only need to store the support vectors and not the entire dataset.

On the other hand, SVMs can be sensitive to the choice of kernel function and the parameters of the algorithm. It is also important to note that SVMs are not suitable for large datasets as the training time can be quite long.

In conclusion, Support Vector Machines (SVMs) are a powerful supervised learning algorithm that can be used for classification and regression problems, especially when the data is not linearly separable. The algorithm is known for its good performance in high-dimensional spaces and its ability to find non-linear boundaries. However, it can be sensitive to the choice of kernel function and parameters, and also not suitable for large datasets.

4.1 Pros

1. Effective in high-dimensional spaces : SVMs have good performance even when the number of features is greater than the number of samples.

2. Memory-efficient : SVMs only need to store the support vectors and not the entire dataset, making them memory-efficient.

3. Versatile : SVMs can be used for both classification and regression problems, and can handle non-linearly separable data using kernel trick.

4. Robust to noise and outliers : SVMs are robust to noise and outliers in the data, as they only rely on the support vectors.

4.2 Cons

1. Sensitive to the choice of kernel function and parameters : The performance of an SVM can be highly dependent on the choice of kernel function and the parameters of the algorithm.

2. Not suitable for large datasets : The training time for SVMs can be quite long for large datasets.

3. Difficulty in interpreting results : It can be difficult to interpret the results of an SVM, especially when using non-linear kernels.

4. Doesn't work well with overlapping classes : SVMs can struggle when classes have significant overlap.

In conclusion, SVMs are a powerful and versatile machine learning algorithm that can be used for both classification and regression problems, especially when the data is not linearly separable. However, they can be sensitive to the choice of kernel function and parameters, not suitable for large datasets, and difficult to interpret the results.

5 Decision tree

Decision trees are a type of machine learning algorithm used for both classification and regression tasks. They are a powerful tool for decision making and can be used to model complex relationships between variables.

A decision tree is a tree-like structure, with each internal node representing a decision point, and each leaf node representing a final outcome or prediction. The tree is built by recursively splitting the data into subsets based on the values of the input features. The goal is to find splits that maximize the separation between the different classes or target values.

One of the main advantages of decision trees is that they are easy to understand and interpret. The tree structure allows for a clear visualization of the decision-making process, and the importance of each feature can be easily assessed.

The process of building a decision tree begins with selecting the root node, which is the feature that best separates the data into different classes or target values. The data is then split into subsets based on the values of this feature, and the process is repeated for each subset until a stopping criterion is met. The stopping criterion can be based on the number of samples in the subsets, the purity of the subsets, or the depth of the tree.

In order to split the nodes at the most informative features, we define an objective function that we want to optimize via the tree learning algorithm. Here, our objective function is to maximize the IG (information gain) at each split : $IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$ where f is the feature to perform the split ; D_p and D_j are the dataset of the parent and j^{th} child node ; I is our impurity measure ; N_p is the total number of training examples at the parent node ; and N_j is the number of examples in the j^{th} child node.

Note that most libraries (including scikit-learn) implrement binary decision trees (this means that each parent node is split into two child nodes, D_{left} and D_{right}) : $IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$.

The three impurity measures or splitting criteria that are commonly used in binary decision trees are **Gini impurity** (I_G), **entropy** (I_H), and the **classification error** (I_E).

- For all non-empty classes, $I_H(t) = -\sum_{i=1}^{c} p(i|t) log_2 p(i|t)$ where $p(i|t)$ is the proportion of the examples that belong to class i for a particular node, t.

- The Gini impurity can be understood as criterion to minimize the probability of misclas-

sification : $I_G(t) = 1 - \sum_{i=1}^{c} p(i|t)^2$.
- $I_E(t) = 1 - \max p(i|t)$.

```
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier(criterion='gini',
                                    max_depth=4,
                                    random_state=1)
...
```

One of the main disadvantages of decision trees is that they can easily overfit the data, particularly when the tree is deep and has many leaves. Overfitting occurs when the tree is too complex and fits the noise in the data rather than the underlying patterns. This can lead to poor generalization performance on new, unseen data. To prevent overfitting, techniques such as pruning, regularization, and cross-validation can be used.

Another problem with decision trees is that they are sensitive to the order of the input features. Different feature orders can lead to different tree structures, and the final tree may not be the optimal one. To overcome this problem, techniques such as random forests and gradient boosting can be used.

In conclusion, decision trees are a powerful and versatile tool for decision-making and predictive modeling. They are easy to understand and interpret, but they can easily overfit the data. To overcome these limitations, various techniques such as pruning, regularization, cross-validation, random forests, and gradient boosting have been developed.

5.1 Pros

1. Easy to understand and interpret : The tree structure allows for a clear visualization of the decision-making process, and the importance of each feature can be easily assessed.

2. Handle both numerical and categorical data : Decision trees can handle both numerical and categorical data, making them a versatile tool for a wide range of applications.

3. High accuracy : Decision trees can achieve high accuracy on many datasets, especially when the tree is not deep.

4. Robust to outliers : Decision trees are not affected by outliers, which makes them suitable for datasets with noise.

5. Can be used for both classification and regression tasks.

5.2 Cons

1. Overfitting : Decision trees can easily overfit the data, particularly when the tree is deep and has many leaves.

2. Sensitive to the order of the input features : Different feature orders can lead to different tree structures, and the final tree may not be the optimal one.

3. Unstable : Decision trees are sensitive to small changes in the data, which can lead to different tree structures and different predictions.

4. Bias : Decision trees can be biased towards features with more levels or categorical variables with many levels, which can lead to inaccurate predictions.

5. Not good for continuous variable : Decision Trees are not good for continuous variable, if the variable is continuous then it could lead to split the variable into many levels, which will make the tree complex and lead to overfitting.

6 Random forest

Random Forest is an ensemble machine learning algorithm that is used for both classification and regression tasks. It is a combination of multiple decision trees, where each tree is grown using a random subset of the data and a random subset of the features. The final prediction is made by averaging the predictions of all the trees in the forest. The idea behind a random forest is to average multiple (deep) decision trees that individually suffer from high variance to build a more robust model that has a better generalization performance and is less susceptible to overfitting. The random forest algorithm can be summarized in four simple steps :
- Draw a random **bootstrap** sample of size n (randomly choose n examples from the training dataset with replacement)
- Grow a decision tree from the bootstrap sample. At each node :
a. Randomly select d features without replacement.
b. Split the node using the feature that provides the best split according to the objective function, for instance, maximizing the information gain.
- Repeat the steps
- Aggregate the prediction by each tree to assign the class label by **majority vote**.
The idea behind using multiple decision trees is that while a single decision tree may be prone to overfitting, a collection of decision trees, or a forest, can reduce the risk of overfitting and improve the overall accuracy of the model.

The process of building a Random Forest begins with creating multiple decision trees using a technique called **bootstrapping**. Bootstrapping is a statistical method that involves randomly selecting data points from the original dataset with replacement. This creates multiple datasets, each with a different set of data points, which are then used to train individual decision trees.

Another important aspect of Random Forest is the use of a random subset of features for each tree. This is known as **random subspace method**. This reduces the correlation between the trees in the forest, which in turn improves the overall performance of the model.

One of the main advantages of Random Forest is that it is less prone to overfitting than a single decision tree. The averaging of multiple trees smooths out the errors and reduces the variance. Random Forest also performs well in high-dimensional datasets and datasets with a large number of categorical variables.

An other advantage of the decision tree algorithm is that it does not require any transformation of the features if we are dealing with nonlinear data, because decision trees analyze one feature at a time, rather than taking weighted combinations into account.
When we used decision trees for classification, we define entropy as a measure of impurity to determine which feature split maximizes the **information gain IG**, which can be defined as follows for a binary split :

$IG(D_p, x_i) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$ where x_i is the feature to perform the split, N_p is the number of training examples in the parent node, I is the impurity function, D_p is the subset of training examples at the parent node, and D_{left}, D_{right} are the subsets of training examples at the left and right child nodes after the split.
To use a decision tree for regression, however, we need an impurity metric that is suitable for continuous variables, so we define the impurity measure of a node, t, as the MSE instead : $I(t) = MSE(t) = \frac{1}{N_t} \sum_{i \in D_t} (y^{(i)} - \hat{y}_t)^2$ where N_t is the number of training examples at node t, D_t is the training subset at node t, $y^{(i)}$ is the true target value, and $\hat{y}_t$ is the predicted target value (sample mean) : $\hat{y}_t = \frac{1}{N_t} \sum_{i \in D_t} y^{(i)}$.
A random forest usually has a better generalization performance than an individual decision tree due to randomness, which helps to decrease the model's variance. Furthermore, they are less sensitive to outliers in the dataset and don't require much parameter tuning. The only parameter in random forests that we typically need to experiment with is the number of trees in the ensemble.

The disadvantage of Random Forest is that it can be computationally expensive to train and make predictions. As the number of trees in the forest increases, the computational time increases as well. Additionally, Random Forest can be less interpretable than a single decision tree because it is harder to understand the contribution of each feature to the final prediction.

In conclusion, Random Forest is a powerful ensemble machine-learning algorithm that can improve the accuracy of decision trees. It is less prone to overfitting and performs well in high-dimensional and categorical datasets. However, it can be computationally expensive and less interpretable than a single decision tree.

7 Naive Bayes

Naive Bayes is a simple and efficient machine learning algorithm that is based on Bayes' theorem and is used for classification tasks. It is called "naive" because it makes the assumption that all the features in the dataset are independent of each other, which is not always the case in real-world data. Despite this assumption, Naive Bayes has been found to perform well in many practical applications.

The algorithm works by using Bayes' theorem to calculate the probability of a given class, given the values of the input features. Bayes' theorem states that the probability of a hypothesis (in this case, the class) given some evidence (in this case, the feature values) is proportional to the probability of the evidence given the hypothesis, multiplied by the prior probability of the hypothesis.

Naive Bayes algorithm can be implemented using different types of probability distributions such as Gaussian, Multinomial, and Bernoulli. Gaussian Naive Bayes is used for continuous data, Multinomial Naive Bayes is used for discrete data, and Bernoulli Naive Bayes is used for binary data.

One of the main advantages of Naive Bayes is its simplicity and efficiency. It is easy to implement and requires less training data than other algorithms. It also performs well on high-dimensional datasets and can handle missing data.

The main disadvantage of Naive Bayes is the assumption of independence between features, which is often not true in real-world data. This can lead to inaccurate predictions, especially when the features are highly correlated. Additionally, Naive Bayes is sensitive to the presence of irrelevant features in the dataset, which can decrease its performance.

In conclusion, Naive Bayes is a simple and efficient machine learning algorithm that is based on Bayes' theorem and is used for classification tasks. It performs well on high-dimensional datasets and can handle missing data but it's main disadvantage is the assumption of independence between features which can lead to inaccurate predictions if the data is not independent.

8 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple and powerful algorithm for classification and regression tasks in machine learning. It is based on the idea that similar data points tend to have similar target values. The algorithm works by finding the k-nearest data points to a given input and using the majority class or average value of the nearest data points to make a prediction.

The process of building a KNN model begins with selecting a value for k, which is the number of nearest neighbors to consider for the prediction. The data is then split into training and test sets, with the training set used to find the nearest neighbors. To make a prediction for a new input, the algorithm calculates the distance between the input and each data point in the training set, and selects the k-nearest data points. The majority class or average value of the nearest data points is then used as the prediction.

One of the main advantages of KNN is its simplicity and flexibility. It can be used for both classification and regression tasks and does not make any assumptions about the underlying data distribution. Additionally, it can handle high-dimensional data and can be used for both supervised and unsupervised learning.

The main disadvantage of KNN is its computational complexity. As the size of the dataset increases, the time and memory required to find the nearest neighbors can become prohibitively large. Additionally, KNN can be sensitive to the choice of k, and finding the optimal value for k can be difficult.

In conclusion, K-Nearest Neighbors (KNN) is a simple and powerful algorithm for classification and regression tasks in machine learning. It is based on the idea that similar data points tend to have similar target values. The main advantage of KNN is its simplicity and flexibility, it can handle high-dimensional data and can be used for both supervised and unsupervised learning. The main disadvantage of KNN is its computational complexity, and it can be sensitive to the choice of k.

9 K-means

K-means is an unsupervised machine-learning algorithm used for clustering. Clustering is the process of grouping similar data points together. K-means is a centroid-based algorithm, or distance-based algorithm, where we calculate the distances to assign a point to a cluster.

The algorithm works by randomly selecting k centroids, where k is the number of clusters we want to form. Each data point is then assigned to the cluster with the nearest centroid. Once all the points have been assigned, the centroids are recalculated as the mean of all the data points in the cluster. This process is repeated until the centroids no longer move or the assignment of points to clusters no longer changes.

One of the main advantages of K-means is its simplicity and scalability. It is easy to implement and can handle large datasets efficiently. Additionally, it is a fast and robust algorithm and it has been widely used in many applications such as image compression, market segmentation, and anomaly detection.

The main disadvantage of K-means is that it assumes that the clusters are spherical and equally sized, which is not always the case in real-world data. Additionally, it is sensitive to the initial placement of centroids and the choice of k. It also assumes that the data is numerical and if the data is not numerical it must be transformed before using the algorithm.

In conclusion, K-means is an unsupervised machine learning algorithm used for clustering. It is based on the idea that similar data points tend to be close to each other. The main advantage of K-means is its simplicity, scalability and it's widely used in many applications. The main disadvantage of K-means is that it assumes that the clusters are spherical and equally sized, it is sensitive to the initial placement of centroids and the choice of k and it assumes that the data is numerical.

10 Dimensionality reduction algorithms

Dimensionality reduction is a technique used to reduce the number of features in a dataset while maintaining the important information. It is used to improve the performance of machine learning algorithms and make data visualization easier. There are several dimensionality reduction algorithms available, including Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and t-Distributed Stochastic Neighbor Embedding (t-SNE).

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that uses orthogonal transformation to convert a set of correlated variables into a set of linearly uncorrelated variables called principal components. PCA is useful for identifying patterns in data and reducing the dimensionality of the data without losing important information.

Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique that is used to find the most discriminative features for the classification task. LDA maximizes the separation between the classes in the lower-dimensional space.

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear dimensionality reduction technique that is particularly useful for visualizing high-dimensional data. It uses probability distributions over pairs of high-dimensional data points to find a low-dimensional representation that preserves the structure of the data.

One of the main advantages of dimensionality reduction techniques is that they can improve the performance of machine learning algorithms by reducing the computational cost and reducing the risk of overfitting. Additionally, they can make data visualization easier by reducing the number of dimensions to a more manageable number.

The main disadvantage of dimensionality reduction techniques is that they can lose important information in the process of reducing the dimensionality. Additionally, the choice of dimensionality reduction technique depends on the type of data and the task at hand, and it can be difficult to determine the optimal number of dimensions to retain.

In conclusion, Dimensionality reduction is a technique used to reduce the number of features in a dataset while maintaining the important information. There are several dimensionality reduction algorithms available such as PCA, LDA and t-SNE which are useful for identifying patterns in data, improving the performance of machine learning algorithms and making data visualization easier. However, it can lose important information in the process of reducing the dimensionality and the choice of dimensionality reduction technique depends on the type of data and the task at hand.

11 Gradient boosting algorithm and AdaBoosting algorithm

Gradient boosting and AdaBoost are two popular ensemble machine learning algorithms that are used for both classification and regression tasks. Both algorithms work by combining multiple weak models to create a strong, final model.

Gradient boosting is an iterative algorithm that builds a model in a forward stage-wise fashion. It starts by fitting a simple model, such as a decision tree, to the data and then adds additional models to correct the errors made by the previous models. Each new model is fit to the negative gradient of the loss function with respect to the previous model's predictions. The final model is a weighted sum of all the individual models.

AdaBoost, short for Adaptive Boosting, is a similar algorithm that also builds a model in a forward stage-wise fashion. However, it focuses on improving the performance of the weak models by adjusting the weights of the training data. In each iteration, the algorithm focuses on the training examples that were misclassified by the previous model, and it adjusts the weights of these examples so that they have a higher probability of being selected in the next iteration. The final model is a weighted sum of all the individual models.

Both gradient boosting and AdaBoost have been found to produce highly accurate models in many practical applications. One of the main advantages of both algorithms is that they can handle a wide range of data types, including categorical and numerical data. Additionally, both algorithms can handle data with missing values, and they are robust to outliers.

One of the main disadvantages of both algorithms is that they can be computationally expensive, especially when the number of models in the ensemble is large. Additionally, they can be sensitive to the choice of the base model and the learning rate.

In conclusion, Gradient boosting and AdaBoost are two popular ensemble machine learning algorithms that are used for both classification and regression tasks. Both algorithms work by combining multiple weak models to create a strong, final model. Both have been found to produce highly accurate models in many practical applications but they can be computationally expensive and sensitive to the choice of the base model and the learning rate.