**What are the two most common supervised tasks?**

Two of the most common supervised learning tasks in machine learning are:

1. **Classification:** This task involves predicting a class label or category for a given input based on past observations.

2. **Regression:** Regression tasks involve predicting continuous numerical values within a given range.

**Can you name four common unsupervised tasks?**

1. **Clustering**: Clustering involves grouping similar data points together based on certain features or characteristics without any predefined labels.

2. **Dimensionality Reduction:** This task involves reducing the number of random variables or features in a dataset while preserving its important structure and relationships.

3. **Anomaly Detection:** Anomaly detection focuses on identifying patterns that do not conform to expected behaviour within a dataset.

4. **Association Rule Learning:** Association rule learning discovers interesting relationships or associations between variables in large datasets.

**Which Linear Regression training algorithm can you use if you have a training set with millions of features?**

1. **Gradient Descent-based methods:** Gradient Descent is particularly useful for large datasets with numerous features. Variants like Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent, or even specialized algorithms like L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) optimization can efficiently handle large feature sets.

2. **Regularization techniques:** Ridge Regression (L2 regularization) or Lasso Regression (L1 regularization) can be employed to handle a large number of features by introducing a penalty term to the cost function. They help in preventing overfitting and can automatically perform feature selection by shrinking less important features' coefficients towards zero.

3. **Feature selection and extraction:** Dimensionality reduction techniques like PCA (Principal Component Analysis) or other feature selection methods such as Recursive Feature Elimination (RFE) can be employed to reduce the number of features before training the linear regression model. This can help in dealing with the curse of dimensionality and improving computational efficiency.

4. **Parallelization and distributed computing:** Implementations that leverage parallel processing or distributed computing frameworks, like Spark's MLlib, can handle large datasets by distributing the computation across multiple machines or cores, making it feasible to process millions of features.

**Suppose the features in your training set have very different scales. Which algorithms might suffer from this, and how? What can you do about it**?

1. **K-Nearest Neighbours (KNN):** Since KNN relies on computing distances between data points, features with different scales can lead to inaccuracies. Features with larger scales can dominate the distance computations, affecting the algorithm's performance.

2. **Gradient Descent-based algorithms:** Algorithms like Linear Regression, Logistic Regression, Neural Networks, and Support Vector Machines (SVM) that use gradient descent for optimization can be adversely affected by varying feature scales. Features with larger scales might have a more considerable impact on the updates of the model parameters, leading to slow convergence or difficulties in reaching the optimal solution.

**Methods to address the issue of varying feature scales:**

1. **Feature Scaling:**

* **Normalization (Min-Max Scaling):** Rescales features to a range between 0 and 1.
* **Standardization (Z-score Normalization):** Transforms features to have a mean of 0 and a standard deviation of 1.

2. **Robust Scaler:**

- This method uses the median and interquartile range to scale features, making it robust to outliers present in the dataset.

3. **Log Transformation:**

- For skewed distributions, applying logarithmic transformation can help normalize the features and mitigate the impact of varying scales.

4. **Normalization within specific algorithms:**

- Some algorithms, like SVMs or neural networks, might benefit from feature scaling for better convergence. Additionally, some implementations or libraries have options to automatically handle feature scaling within the algorithms.

**Do all Gradient Descent algorithms lead to the same model, provided you let them run long enough?**

No, not all Gradient Descent algorithms will necessarily lead to the exact same model, even if allowed to run for a long time. While various Gradient Descent-based optimization algorithms aim to minimize the loss function, differences in their update rules, convergence criteria, and step size adjustments can lead to different paths and potentially different solutions or models.

**Suppose you use Batch Gradient Descent and you plot the validation error at every epoch. If you notice that the validation error consistently goes up, what is likely going on? How can you fix this?**

If the validation error consistently increases (rather than decreases or fluctuates) at every epoch during training using Batch Gradient Descent, it suggests that the model is not generalizing well to unseen data. This scenario indicates overfitting, where the model learns to perform well on the training data but fails to generalize to new, unseen data from the validation set.

**Potential strategies to address increasing validation error:**

1. **Regularization Techniques:**

* **L2 (Ridge) or L1 (Lasso) regularization:** Introduce penalty terms on the model's weights to prevent overfitting by reducing their magnitudes.
* **Early Stopping:** Monitor the validation error during training and stop the training process when the validation error starts to increase consistently, thereby preventing overfitting.

2. **Reduce Model Complexity:**

* **Feature Selection/Dimensionality Reduction:** Remove irrelevant or redundant features or use techniques like PCA to reduce the number of features.
* **Simplify the Model Architecture:** Use a simpler model with fewer layers or parameters to reduce complexity.

3. **Increase Training Data:**

Collect more data if possible. A larger, more diverse dataset can help the model to generalize better.

4. **Adjust Hyperparameters:**

Tune hyperparameters such as learning rate, batch size, or regularization strength. This might involve performing a grid search or using techniques like Bayesian optimization to find optimal values.

**Suppose you are using Ridge Regression and you notice that the training error and the validation error are almost equal and fairly high. Would you say that the model suffers from high bias or high variance?**

If both the training error and validation error in Ridge Regression are almost equal and relatively high, it suggests that the model suffers from high bias rather than high variance because,

High bias occurs when the model is too simple to capture the underlying patterns in the data. It leads to underfitting, where the model cannot even capture the trends present in the training data, resulting in both high training and validation errors that are close to each other and relatively high.

High variance, on the other hand, typically shows a large gap between the training error and the validation error. In this scenario, the model is too complex, capturing noise or specific details in the training data but failing to generalize to unseen data, leading to a significantly lower training error compared to the validation error.

**What is a support vector?**

In the context of Support Vector Machines (SVM), a support vector refers to the data points from the training dataset that are used to define the decision boundary between different classes.

**Can an SVM classifier output a confidence score when it classifies an instance? What about a probability?**

Yes, an SVM classifier can provide a confidence score or estimate probabilities for its predictions, depending on the type of SVM used and whether it incorporates additional techniques.

**Say you’ve trained an SVM classifier with an RBF kernel, but it seems to underfit the training set. Should you increase or decrease γ (gamma)? What about C?**

If your SVM classifier with an RBF (Radial Basis Function) kernel is underfitting the training set, you need to adjust the hyperparameters γ (gamma) and C to potentially improve the model's performance.

**γ (gamma):**

If the model is underfitting, meaning it's too simplistic and unable to capture the complexity of the data, you might need to increase γ as this would make the decision boundary more sensitive to individual data points and could lead to a more complex (less smooth) decision boundary, potentially better fitting the training data.

**C:**

A low C value encourages a larger margin by allowing more misclassifications (soft margin), which might lead to underfitting. If the model is underfitting, you might want to increase C to penalize misclassifications more strongly, potentially fitting the training data better.