12 AI Algorithms

Every Data Analyst Should Know

A Quick-Reference Guide for Interviews and On-the-Job Decisions

For each algorithm, this guide covers:

- What problem it solves
- How it works (in plain language)
- When to use it
- Its key strengths and weaknesses
- What interviewers typically expect you to know
- A ready-to-use Python code example

Contents

1. Linear Regression

Problem it solves	Predicting a continuous numeric value based on the relationship between input variables and an output variable.
How it works	Fits a straight line (or hyperplane) through the data that minimizes the sum of squared differences between predicted and actual values.
When to use it	Forecasting sales, predicting prices, estimating demand, or any task where the output is a number and the relationship with inputs is roughly linear.

Strengths

- Simple to implement and interpret
- Fast to train, even on large datasets
- Coefficients show direction and magnitude of each feature's effect

Weaknesses

- Assumes a linear relationship between inputs and output
- Sensitive to outliers
- Struggles with multicollinearity among features

Interview Tip

Be ready to explain assumptions (linearity, independence, homoscedasticity, normal residuals), how to interpret coefficients, and how R-squared and p-values relate to model fit.

Python Example

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)
print("RMSE:", mean_squared_error(y_test, predictions, squared=False))
print("Coefficients:", model.coef_)
```

2. Logistic Regression

Problem it solves	Predicting a binary or categorical outcome (yes/no, churn/no churn, fraud/not fraud).
How it works	Uses the sigmoid function to convert a linear combination of inputs into a probability between 0 and 1, then applies a threshold to classify.
When to use it	Churn prediction, credit scoring, medical diagnosis (disease/no disease), email spam detection.

Strengths

- Outputs interpretable probabilities
- Computationally efficient
- Performs well when classes are linearly separable

Weaknesses

- Assumes a linear decision boundary
- Struggles with complex, non-linear relationships
- Sensitive to imbalanced classes

Interview Tip

Know the difference between odds, log-odds, and probability, how to choose a classification threshold, and how it differs from linear regression despite the similar name.

Python Example

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, predictions))
```

3. Decision Tree

Problem it solves	Classification or regression tasks where you need a model that's easy to explain and visualize.
How it works	Splits data repeatedly based on feature values to create a tree of decision rules, ending in leaf nodes that represent predictions.
When to use it	Customer segmentation, approval/rejection decisions, any case where stakeholders need to see exactly why a prediction was made.

Strengths

- Easy to visualize and explain to non-technical stakeholders
- Handles both numerical and categorical data
- No need for feature scaling

Weaknesses

- Prone to overfitting, especially with deep trees
- Small changes in data can produce a very different tree
- Generally less accurate than ensemble methods

Interview Tip

Be ready to discuss splitting criteria (Gini impurity, entropy/information gain), pruning, and how depth controls the bias-variance tradeoff.

Python Example

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = DecisionTreeClassifier(max_depth=4, criterion="gini", random_state=42)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
print("Feature importances:", model.feature_importances_)
```

4. Random Forest

Problem it solves	Improving on a single decision tree's accuracy and stability for classification or regression.
How it works	Builds many decision trees on random subsets of data and features, then averages (regression) or votes (classification) across all trees.
When to use it	Risk modeling, feature importance analysis, general-purpose predictive modeling where accuracy matters more than full interpretability.

Strengths

- Reduces overfitting compared to a single tree
- Handles large datasets with high dimensionality well
- Provides built-in feature importance scores

Weaknesses

- Less interpretable than a single decision tree
- Can be slow to train and predict with many trees
- Larger memory footprint

Interview Tip

Understand bagging (bootstrap aggregating), how randomness is introduced (row and feature sampling), and why it reduces variance.

Python Example

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(n_estimators=200, max_depth=None, random_state=42)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
print("Top features:", model.feature_importances_)
```

5. Gradient Boosting

Problem it solves	Achieving high-accuracy predictions for structured/tabular data, classification or regression.
How it works	Builds trees sequentially, where each new tree corrects the errors (residuals) of the combined previous trees, using gradient descent to minimize a loss function.
When to use it	Ranking systems, fraud detection, demand forecasting, and most Kaggle-style tabular competitions.

Strengths

- Often achieves top-tier accuracy on structured data
- Flexible with different loss functions
- Handles mixed data types well

Weaknesses

- Prone to overfitting if not tuned carefully
- Slower to train than random forest (sequential, not parallel)
- Many hyperparameters to tune

Interview Tip

Know the difference between bagging (random forest) and boosting, how learning rate and number of estimators interact, and the concept of residual fitting.

Python Example

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = GradientBoostingClassifier(
    n_estimators=150, learning_rate=0.1, max_depth=3, random_state=42
)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
```

6. XGBoost

Problem it solves	Same as gradient boosting, but optimized for speed, scalability, and competition-grade performance.
How it works	An optimized implementation of gradient boosting with regularization (L1/L2), parallel processing, tree pruning, and built-in handling of missing values.
When to use it	Industry-standard for tabular data problems: credit risk, click-through-rate prediction, sales forecasting, and most structured-data ML competitions.

Strengths

- Faster and more memory-efficient than standard gradient boosting
- Built-in regularization reduces overfitting
- Handles missing data automatically

Weaknesses

- More complex to tune (many hyperparameters)
- Less interpretable as a 'black box' model
- Can still overfit small datasets

Interview Tip

Be prepared to discuss why XGBoost often outperforms plain gradient boosting, regularization parameters (lambda, alpha), and early stopping.

Python Example

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = xgb.XGBClassifier(
    n_estimators=200, learning_rate=0.1, max_depth=4,
    reg_lambda=1.0, reg_alpha=0.0, use_label_encoder=False,
    eval_metric="logloss", random_state=42
)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
```

7. Support Vector Machine (SVM)

Problem it solves	Classification (and some regression) tasks, especially with clear margins between classes.
How it works	Finds the optimal hyperplane that maximizes the margin between classes; kernel functions allow it to handle non-linear boundaries by mapping data to higher dimensions.
When to use it	Text classification, image classification, bioinformatics (gene classification), and other high-dimensional problems with relatively small datasets.

Strengths

- Effective in high-dimensional spaces
- Works well with clear margins of separation
- Kernel trick allows handling non-linear data

Weaknesses

- Doesn't scale well to very large datasets
- Sensitive to choice of kernel and hyperparameters
- Less interpretable than simpler models

Interview Tip

Understand the concept of the margin, support vectors, the kernel trick (linear, polynomial, RBF), and the role of the C parameter.

Python Example

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# SVM is sensitive to scale - always scale features first
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = SVC(kernel="rbf", C=1.0, gamma="scale")
model.fit(X_train_scaled, y_train)

predictions = model.predict(X_test_scaled)
print("Accuracy:", accuracy_score(y_test, predictions))
```

8. k-Nearest Neighbors (KNN)

Problem it solves	Simple classification or regression based on similarity to existing labeled data points.
How it works	For a new data point, finds the 'k' closest points in the training data (by distance metric) and predicts based on their majority class or average value.
When to use it	Recommendation systems, anomaly detection, simple classification tasks where data is well-clustered.

Strengths

- Simple and intuitive, no training phase required
- Naturally handles multi-class problems
- Adapts easily to new data

Weaknesses

- Slow at prediction time on large datasets
- Sensitive to feature scaling and irrelevant features
- Choice of 'k' and distance metric greatly affects results

Interview Tip

Know why feature scaling matters, how to choose 'k' (and the bias-variance tradeoff involved), and common distance metrics (Euclidean, Manhattan).

Python Example

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = KNeighborsClassifier(n_neighbors=5, metric="euclidean")
model.fit(X_train_scaled, y_train)

predictions = model.predict(X_test_scaled)
print("Accuracy:", accuracy_score(y_test, predictions))
```

9. Naive Bayes

Problem it solves	Fast, probabilistic classification, especially for text data.
How it works	Applies Bayes' theorem with the 'naive' assumption that all features are independent of each other, calculating the probability of each class given the input features.
When to use it	Spam filtering, sentiment analysis, document categorization, and other text-classification tasks.

Strengths

- Extremely fast to train and predict
- Performs well even with small training datasets
- Works surprisingly well for text data despite the independence assumption

Weaknesses

- The independence assumption rarely holds in real data
- Can be outperformed by more complex models on non-text data
- Struggles when features are highly correlated

Interview Tip

Be ready to explain Bayes' theorem, why the 'naive' independence assumption is made, and why it still performs well in practice for text.

Python Example

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2,
random_state=42)

vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

model = MultinomialNB()
model.fit(X_train_vec, y_train)

predictions = model.predict(X_test_vec)
print("Accuracy:", accuracy_score(y_test, predictions))
```

10. k-Means Clustering

Problem it solves	Unsupervised grouping of data points into clusters based on similarity, when there are no labels.
How it works	Randomly initializes 'k' cluster centers, assigns each point to the nearest center, then recalculates centers iteratively until assignments stabilize.
When to use it	Customer segmentation, market basket grouping, image compression, and exploratory data analysis to find natural groupings.

Strengths

- Simple, fast, and scalable to large datasets
- Easy to interpret cluster assignments
- Works well when clusters are roughly spherical and similarly sized

Weaknesses

- Requires choosing 'k' in advance
- Sensitive to initial centroid placement and outliers
- Struggles with non-spherical or unevenly sized clusters

Interview Tip

Know how to choose 'k' (elbow method, silhouette score), the impact of initialization, and the assumption of spherical clusters.

Python Example

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

model = KMeans(n_clusters=4, init="k-means++", n_init=10, random_state=42)
clusters = model.fit_predict(X_scaled)

print("Cluster centers:", model.cluster_centers_)
print("Silhouette score:", silhouette_score(X_scaled, clusters))
```

11. Convolutional Neural Networks (CNN)

Problem it solves	Tasks involving image, video, or spatial data where local patterns matter.
How it works	Uses convolutional layers with filters that slide across the input to detect features like edges, textures, and shapes, building up to higher-level representations through multiple layers.
When to use it	Image classification, object detection, facial recognition, and medical image analysis.

Strengths

- Automatically learns relevant features from raw pixel data
- Captures spatial hierarchies and patterns effectively
- State-of-the-art performance on image-related tasks

Weaknesses

- Requires large amounts of labeled data and compute
- Less interpretable ('black box')
- Overkill for simple tabular or structured data problems

Interview Tip

Understand the role of convolutional layers, pooling, and why CNNs are well-suited to grid-like data such as images.

Python Example

```
import tensorflow as tf
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation="relu"),
    layers.Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

12. Transformers

Problem it solves	Tasks involving sequential data, especially natural language, where context and long-range relationships matter.
How it works	Uses a self-attention mechanism that weighs the importance of different parts of the input relative to each other, processing sequences in parallel rather than step-by-step.
When to use it	Language translation, text generation (like chatbots), summarization, and increasingly, time-series and even some vision tasks.

Strengths

- Captures long-range dependencies better than older sequence models (RNNs/LSTMs)
- Highly parallelizable, enabling faster training on large datasets
- Forms the backbone of modern large language models

Weaknesses

- Computationally expensive and resource-intensive
- Requires large amounts of training data to perform well
- Complex architecture that's harder to fully interpret

Interview Tip

Know the basics of self-attention, why transformers replaced RNNs for many NLP tasks, and at a high level how models like GPT and BERT are built on this architecture.

Python Example

```
from transformers import pipeline

# Use a pretrained transformer for sentiment analysis
classifier = pipeline("sentiment-analysis")

results = classifier([
    "I love this product, it works great!",
    "This was a terrible experience."
])

for r in results:
    print(r)
```

Quick Comparison Cheat Sheet

Algorithm	Type	Best For
Linear Regression	Supervised - Regression	Predicting continuous values with linear relationships
Logistic Regression	Supervised - Classification	Binary outcomes (yes/no, churn/no churn)
Decision Tree	Supervised - Both	Explainable rules and simple decision logic
Random Forest	Supervised - Both	Robust, general-purpose predictions
Gradient Boosting	Supervised - Both	High-accuracy tabular data modeling
XGBoost	Supervised - Both	Optimized, large-scale tabular data tasks
SVM	Supervised - Both	High-dimensional data with clear margins
KNN	Supervised - Both	Simple similarity-based predictions
Naive Bayes	Supervised - Classification	Fast text and document classification
k-Means	Unsupervised - Clustering	Finding natural groupings in unlabeled data
CNN	Deep Learning	Image and spatial data
Transformers	Deep Learning	Text, language, and sequence data

Save this guide, share it with your network, and good luck with your next interview!