AI/ML Engineer Interview Questions and Answers

◇ 1. Random Forest Classifier

Conceptual Questions

Q: What is a Random Forest?

A Random Forest is an ensemble learning method that builds multiple decision trees and merges their results to improve accuracy and control overfitting.

Q: How does Random Forest reduce overfitting compared to decision trees?

It reduces overfitting by averaging multiple decision trees trained on different subsets of the data, which lowers variance.

Q: What is the role of bootstrapping in Random Forest?

Bootstrapping is used to create random subsets of the training data with replacement for each decision tree, introducing diversity.

Q: How is feature importance calculated in Random Forest?

Feature importance is typically measured by how much each feature decreases the impurity (Gini or entropy) across all trees.

Q: What's the difference between Random Forest and Bagging?

Bagging uses multiple base models trained on bootstrapped datasets, while Random Forest adds feature-level randomness in addition to bootstrapping.

Practical/Scenario-Based

Q: When would you prefer Random Forest over Gradient Boosting?

When model interpretability and faster training are prioritized, or when the data contains noisy features.

Q: How do you tune hyperparameters in Random Forest (e.g., n_estimators, max_depth)?

Use GridSearchCV or RandomizedSearchCV to tune parameters like n_estimators, max_depth, min_samples_split, etc.

Q: How do you handle imbalanced data with Random Forest?

Use class_weight='balanced', stratified sampling, or resampling techniques like SMOTE.

Q: What are the limitations of Random Forest?

Slower predictions, large model size, less effective on sparse or high-dimensional data compared to boosting methods.

Code-Based

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

rf = RandomForestClassifier()
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20]
}
gs = GridSearchCV(rf, param_grid, cv=5)
gs.fit(X_train, y_train)
print(gs.best_params_)

# Feature importances
import matplotlib.pyplot as plt
import seaborn as sns
importances = gs.best_estimator_.feature_importances_
sns.barplot(x=importances, y=feature_names)
```

◇ 2. XGBoost

Conceptual Questions

Q: What is XGBoost and how does it differ from Random Forest?

XGBoost is a gradient boosting algorithm that builds trees sequentially to correct previous errors, whereas Random Forest builds trees in parallel.

Q: Explain how gradient boosting works.

It builds trees sequentially, each trying to correct the errors of the previous one by fitting to the residuals.

Q: What are some regularization techniques used in XGBoost?

L1 (alpha) and L2 (lambda) regularization to prevent overfitting.

Q: What is the role of learning_rate, gamma, lambda, and alpha in XGBoost?

- learning_rate: Controls the contribution of each tree
- gamma: Minimum loss reduction to make a split
- lambda: L2 regularization
- alpha: L1 regularization

Q: Why is XGBoost often preferred in Kaggle competitions?

It offers high accuracy, supports missing data, fast computation, and extensive tuning options.

Practical/Scenario-Based

Q: How do you avoid overfitting in XGBoost?

Use early stopping, regularization (lambda, alpha), and control depth/number of trees.

Q: How would you perform hyperparameter tuning for XGBoost?

Use RandomizedSearchCV or Optuna with evaluation metrics and early stopping on a validation set.

Q: How does early stopping work in XGBoost?

Stops training if evaluation metric does not improve for a set number of rounds.

Q: What is the difference between LightGBM, XGBoost, and CatBoost?

- LightGBM: Fast, uses histogram-based methods

- XGBoost: Robust, highly customizable
- CatBoost: Handles categorical features natively

Code-Based

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

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
dtrain = xgb.DMatrix(X_train, label=y_train)
dval = xgb.DMatrix(X_val, label=y_val)

params = {'objective': 'binary:logistic', 'eval_metric': 'logloss'}
model = xgb.train(params, dtrain, evals=[(dval, 'validation')],
early_stopping_rounds=10)

# SHAP
import shap
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_val)
shap.summary_plot(shap_values, X_val)
```

◇ 3. LSTM & CNN (Deep Learning)

LSTM - Conceptual Questions

Q: How does an LSTM differ from a vanilla RNN?

LSTMs include memory cells and gating mechanisms to retain long-term dependencies and avoid vanishing gradients.

Q: What are the components of an LSTM cell (forget, input, output gates)?

- Forget Gate: Decides what information to discard
- Input Gate: Decides what new information to store
- Output Gate: Decides what information to output

Q: How does LSTM handle long-term dependencies?

LSTM maintains a cell state that is updated through controlled gates, allowing information to persist over long sequences.

Q: What's the difference between LSTM and GRU?

GRU has fewer gates (update and reset), is simpler and often faster but may perform slightly worse in some contexts.

CNN - Conceptual Questions

Q: What are convolutional layers and how do they work?

Convolutional layers apply filters to input data to extract spatial features using convolution operations.

Q: Explain max pooling and its purpose.

Max pooling reduces dimensionality by selecting the maximum value in a window, improving computational efficiency and invariance.

Q: How does CNN handle spatial hierarchies in images?

Multiple convolution layers extract increasing levels of spatial features, from edges to complex patterns.

Q: What is receptive field and how does it affect learning?

It is the region in the input that a particular neuron can 'see'. Larger receptive fields capture more complex patterns.

Practical/Scenario-Based

Q: How would you use an LSTM for time-series forecasting?

Prepare a windowed dataset, normalize inputs, and train an LSTM to predict future time steps based on past values.

Q: How can CNNs be used in NLP tasks?

Apply 1D convolutions over text embeddings to capture n-gram patterns for classification tasks.

Q: What's the difference between 1D, 2D, and 3D convolutions?

- 1D: Used for sequences (e.g., text)
- 2D: Used for images
- 3D: Used for volumetric or video data

Q: Describe a real-world use case combining CNN and LSTM.

Video classification: CNN extracts spatial features from frames; LSTM models temporal dependencies across frames.

Code-Based

```
# CNN for image classification
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='categorical_crossentropy',
metrics=['accuracy'])

# LSTM for time series forecasting
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential([
    LSTM(50, return_sequences=False, input_shape=(n_timesteps,
n_features)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')

# CNN + LSTM for video classification (example skeleton)
cnn_model = models.Sequential([
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(64, 64,
3)),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten()
])

video_input = tf.keras.Input(shape=(sequence_length, 64, 64, 3))
x = layers.TimeDistributed(cnn_model)(video_input)
x = LSTM(50)(x)
output = Dense(10, activation='softmax')(x)
video_model = tf.keras.Model(inputs=video_input, outputs=output)
video_model.compile(optimizer='adam', loss='categorical_crossentropy')
```

◇ 4. Model Evaluation & Tuning

Conceptual Questions

Q: What are common evaluation metrics for classification?

Accuracy, Precision, Recall, F1-Score, ROC-AUC, Log Loss.

Q: What is the difference between precision and recall?

Precision is the ratio of true positives to predicted positives. Recall is the ratio of true positives to actual positives.

Q: What is cross-validation and why is it used?

Cross-validation is a resampling method to evaluate model performance on different data splits to reduce overfitting and bias.

Q: What is the purpose of a confusion matrix?

To show the count of true positives, true negatives, false positives, and false negatives in classification.

Q: How do you choose between precision and recall?

Choose precision when false positives are costly; recall when false negatives are costly (e.g., spam detection vs. disease diagnosis).

Q: What is ROC-AUC and why is it useful?

ROC-AUC measures a classifier’s ability to distinguish between classes; higher values mean better distinction.

Practical/Scenario-Based

Q: How do you handle class imbalance during model evaluation?

Use stratified sampling, precision-recall curves, AUC-PR, class-weighting, and resampling methods.

Q: What is the difference between GridSearchCV and RandomizedSearchCV?

GridSearchCV tests all parameter combinations, while RandomizedSearchCV samples a given number randomly from parameter space, making it faster.

Q: When would you use early stopping?

To prevent overfitting by halting training when validation performance degrades.

Q: How do you validate a time-series model?

Use techniques like TimeSeriesSplit or walk-forward validation instead of random k-fold splits.

Q: What are some tips for tuning hyperparameters?

Start with coarse ranges, use domain knowledge, prioritize impactful parameters (e.g., learning_rate, max_depth), and automate with Optuna or Bayesian optimization.

Code-Based

```
# Classification metrics
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score

y_pred = model.predict(X_test)
y_pred_class = y_pred > 0.5
print(classification_report(y_test, y_pred_class))
print(confusion_matrix(y_test, y_pred_class))
print("ROC-AUC:", roc_auc_score(y_test, y_pred))

# Cross-validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
print("CV Accuracy:", scores.mean())

# Grid Search Example
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth': [3, 5, 7], 'learning_rate': [0.01, 0.1]}
gs = GridSearchCV(estimator, param_grid, cv=3)
gs.fit(X_train, y_train)
print(gs.best_params_)
```