Machine Learning Certification Exam

Machine Learning Certification Exam

This questionnaire consists of 60 questions in a "complete the code" format, covering key machine learning concepts and their implementation in Python with scikit-learn. Each question requires filling in the blank to complete the code or concept.

Section 1: Machine Learning Basics (Questions 1–10)

1. Import the main machine learning library in Python:
   `import __ as skl`
2. Define machine learning:
   `def definicion_ml():`
   `return "Machine learning is a subfield of AI that enables machines to learn from __ without explicit programming."`
3. List a machine learning application:
   `aplicaciones = ["Netflix recommendations", "Fraud detection", "__ in medicine"]`
4. Complete the typical stages of a machine learning problem:
   `etapas = ["Data collection", "Preprocessing", "Model training", "__", "Deployment"]`
5. List types of machine learning algorithms:
   `tipos_algoritmos = ["Supervised", "Unsupervised", "By __", "Semi-supervised"]`
6. Compare supervised vs. unsupervised learning:
   `supervisado = "Uses labeled data to predict"`
   `no_supervisado = "Finds patterns in __ data"`
7. Define a regression task:
   `def tarea_regresion():`
   `return "Predict __ continuous values, like house prices."`
8. List commonly used regression algorithms:
   `algoritmos_regresion = ["Linear Regression", "Decision Trees", "__"]`
9. Types of classification tasks:
   `tipos_clasificacion = ["Binary:  two classes", "Multiclass:  more than two classes", "Multilabel:  __ per instance"]`

10. Explain the curse of dimensionality:
```
def maldicion_dim():
return "As __ increase, data becomes sparse, reducing model performance."
```

Section 2: Data Preprocessing (Questions 11–25)

11. Import the preprocessing module from scikit-learn:
```
from sklearn.__ import LabelEncoder
```
12. Apply Label Encoding:
```
encoder = __()
encoded = encoder.fit_transform(categorias)
```
13. Apply One-Hot Encoding:
```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
one_hot = encoder.__(datos_categ)
```
14. Create dummy variables with pandas:
```
import pandas as pd
df_dummy = pd.__(df, columns=['categoria'])
```
15. Implement Min-Max scaling:
```
from sklearn.preprocessing import MinMaxScaler
scaler = __()
datos_escalados = scaler.fit_transform(datos)
```
16. Implement Standard scaling:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
datos = scaler.__(X)
```
17. Define Euclidean distance:
```
import numpy as np
def euclidiana(a, b):
return np.sqrt(np.sum((a - b)**__))
```
18. Define Manhattan distance:
```
def manhattan(a, b):
return np.sum(np.__(a - b))
```
19. Define Minkowski distance:
```
def minkowski(a, b, p):
return np.sum(np.abs(a - b)**p)**(1/__)
```
20. List common preprocessing tasks:
```
tareas = ["Data cleaning", "Handling missing values", "__ of features"]
```
21. One-Hot Encoding with scikit-learn:
```
encoder = OneHotEncoder(sparse=__)
encoded = encoder.fit_transform(df[['columna']])
```
22. Dummy variables avoiding multicollinearity:
```
df_dummy = pd.get_dummies(df, columns=['categoria'], drop_first=__)
```

23. Explain the concept of distance:
```
concepto = "The __ measures similarity; scaling ensures equal feature
contribution."
```
24. Min-Max scaler range:
```
scaler = MinMaxScaler(feature_range=(0, __))
```
25. StandardScaler configuration:
```
scaler = StandardScaler(with_mean=True, with_std=__)
```

Section 3: Regression and Classification Algorithms (Questions 26–60)

26. Import logistic regression:
```
from sklearn.linear_model import __
```
27. Define the sigmoid function:
```
def sigmoide(z):
return 1 / (1 + np.exp(__z))
```
28. Advantages of logistic regression:
```
ventajas = ["Easy interpretation", "Probability outputs", "__ for binary
classification"]
```
29. Disadvantages of logistic regression:
```
desventajas = ["Assumes linearity", "Sensitive to __"]
```
30. Fit logistic regression:
```
model = LogisticRegression()
model.__(X_train, y_train)
```
31. Import KNN classifier:
```
from sklearn.neighbors import __
```
32. KNN with k=5:
```
knn = KNeighborsClassifier(n_neighbors=__)
```
33. Scaling for KNN:
```
razon = "Scaling prevents large-range features from dominating the __."
```
34. Euclidean distance in KNN:
```
knn = KNeighborsClassifier(metric='__')
```
35. Selecting k in KNN:
```
def seleccionar_k():
return "Use cross-validation to maximize __."
```
36. Import decision tree:
```
from sklearn.tree import __
```
37. Decision tree hyperparameters:
```
tree = DecisionTreeClassifier(max_depth=5, min_samples_split=__)
```
38. Impurity measures:
```
impureza = ["Gini", "__"]
```

39. Advantages of decision trees:
`ventajas = ["Easy visualization", "Handles non-linear data", "__ preprocessing required"]`

40. Disadvantages of decision trees:
`desventajas = ["Prone to overfitting", "__ predictions"]`

41. Fit decision tree:
```
tree = DecisionTreeClassifier()
tree.fit(X, __)
```

42. Import random forest:
`from sklearn.ensemble import __`

43. Bagging concept:
`bagging = "Train multiple models on __ subsets and average predictions."`

44. Random forest configuration:
`forest = RandomForestClassifier(n_estimators=__)`

45. Import SVM classifier:
`from sklearn.svm import __`

46. How SVM works:
`svm = "Finds a hyperplane that maximizes the __ between classes."`

47. SVM problem type:
`problema = "Classification __, especially with high-dimensional data."`

48. SVM kernel types:
`kernels = ["Linear", "Polynomial", "__", "Sigmoid"]`

49. RBF kernel characteristics:
`rbf = "Maps to infinite space, good for __ data."`

50. SVM with linear kernel:
`svm = SVC(kernel='__')`

51. Logistic regression for multiclass:
`model = LogisticRegression(multi_class='__')`

52. KNN for regression:
```
from sklearn.neighbors import KNeighborsRegressor
knn_reg = __(n_neighbors=3)
```

53. Decision tree impurity criterion:
`tree = DecisionTreeClassifier(criterion='__')`

54. Bagging in random forest:
`forest = RandomForestClassifier(bootstrap=__)`

55. Polynomial kernel in SVM:
`svm = SVC(kernel='poly', degree=__)`

56. SVM advantages:
`ventajas = ["Effective in high dimensionality", "__ in memory"]`

57. SVM disadvantages:
```
desventajas = ["Slow on large datasets", "Sensitive to __ choice"]
```
58. Compute accuracy in KNN:
```
from sklearn.metrics import accuracy_score
acc = __(y_test, preds)
```
59. Prevent overfitting in decision tree:
```
tree = DecisionTreeClassifier(max_depth=__)
```
60. SVM for regression:
```
from sklearn.svm import SVR
svr = __(kernel='rbf')
```