Lista 4 de IA Giusppe Cordeiro

Questao 1

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from yellowbrick.classifier import ConfusionMatrix
from skopt import BayesSearchCV

def load_data():
    training_data = pd.read_csv('titanic/train.csv')
    test_data = pd.read_csv('titanic/test.csv')
    truth_table = pd.read_csv('titanic/gender_submission.csv')
    test_data = test_data.merge(truth_table, on='PassengerId', how='left')
    return training_data, test_data

def preprocess_data(training_data, test_data):
    drop_cols = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'Embarked']
    encoder = LabelEncoder()
    training_data['Sex'] = encoder.fit_transform(training_data['Sex'])
    test_data['Sex'] = encoder.transform(test_data['Sex'])

    training_data['Age'] = training_data['Age'].fillna(0)
    test_data['Age'] = test_data['Age'].fillna(0)
    training_data['Fare'] = training_data['Fare'].fillna(99999)
    test_data['Fare'] = test_data['Fare'].fillna(99999)

    X_treino = training_data.drop(columns=drop_cols + ['Survived'], axis=1)
    y_treino = training_data['Survived']
    X_teste = test_data.drop(columns=drop_cols + ['Survived'], axis=1)
    y_teste = test_data['Survived']

    return X_treino, y_treino, X_teste, y_teste

def optimize_hyperparameters(X_treino, y_treino):
    dt_params = {
        'max_depth': (None, 1, 2, 3),
        'max_features': ['sqrt', 'log2', None],  # Ajustado, 'auto' foi removido
        'min_samples_split': (100, 200)
    }
    dt_model = BayesSearchCV(
        DecisionTreeClassifier(),
        search_spaces=dt_params,
        cv=5,
        n_jobs=-1,
        verbose=1
    )
    dt_model.fit(X_treino, y_treino)

    rf_model = BayesSearchCV(
        RandomForestClassifier(),
        search_spaces=dt_params,
        cv=5,
        n_jobs=-1,
        verbose=1
    )
    rf_model.fit(X_treino, y_treino)
    return dt_model.best_params_, rf_model.best_params_

def train_decision_tree(X_treino, y_treino, dt_params):
    dt_model = DecisionTreeClassifier(**dt_params)  # Usa os parâmetros ajustados
    dt_model.fit(X_treino, y_treino)
    return dt_model

def train_random_forest(X_treino, y_treino, rf_params):
    rf_model = RandomForestClassifier(**rf_params)  # Usa os parâmetros ajustados
    rf_model.fit(X_treino, y_treino)
    return rf_model

def evaluate_model(model, X_teste, y_teste, model_name):
    predictions = model.predict(X_teste)
    print(f"Acurácia - {model_name}:", accuracy_score(y_teste, predictions))
    print(f"Matriz de Confusão - {model_name}:\n", confusion_matrix(y_teste, predictions))
    print(f"Relatório de Classificação - {model_name}:\n", classification_report(y_teste, predictions))

def plot_feature_importance(model, X_treino, model_name):
    if hasattr(model, 'feature_importances_'):
```

```
    if hasattr(model, 'feature_importances_'):
        importancias = model.feature_importances_
        features = pd.DataFrame({'Feature': X_treino.columns, 'Importância': importancias})
        print(f"Importância dos Atributos - {model_name}")
        print(features)

def plot_confusion_matrix(model, X_treino, y_treino, X_teste, y_teste, model_name):
    cm = ConfusionMatrix(model)
    cm.fit(X_treino, y_treino)
    cm.score(X_teste, y_teste)
    cm.show()


training_data, test_data = load_data()
X_treino, y_treino, X_teste, y_teste = preprocess_data(training_data, test_data)
dt_params, rf_params = optimize_hyperparameters(X_treino, y_treino)
dt_model = train_decision_tree(X_treino, y_treino, dt_params)
rf_model = train_random_forest(X_treino, y_treino, rf_params)
evaluate_model(dt_model, X_teste, y_teste, "Decision Tree")
evaluate_model(rf_model, X_teste, y_teste, "Random Forest")
plot_feature_importance(rf_model, X_treino, "Random Forest")
```

Questao 2

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from yellowbrick.classifier import ConfusionMatrix
from imblearn.over_sampling import SMOTE
from skopt import BayesSearchCV
from sklearn import tree

def load_data():
    training_data = pd.read_csv('titanic/train.csv')
    test_data = pd.read_csv('titanic/test.csv')
    truth_table = pd.read_csv('titanic/gender_submission.csv')
    test_data = test_data.merge(truth_table, on='PassengerId', how='left')
    return training_data, test_data

def preprocess_data(training_data, test_data):
    columns_to_drop = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'Embarked']
    encoder = LabelEncoder()
    training_data['Sex'] = encoder.fit_transform(training_data['Sex'])
    test_data['Sex'] = encoder.transform(test_data['Sex'])

    for col in ['Age', 'Fare']:
        training_data[col] = training_data[col].fillna(training_data[col].median())
        test_data[col] = test_data[col].fillna(test_data[col].median())

    X_treino = training_data.drop(columns=columns_to_drop + ['Survived'], axis=1)
    y_treino = training_data['Survived']
    X_teste = test_data.drop(columns=columns_to_drop + ['Survived'], axis=1)
    y_teste = test_data['Survived']

    return X_treino, y_treino, X_teste, y_teste

def balance_data(X_treino, y_treino):
    smote = SMOTE(random_state=42)
    return smote.fit_resample(X_treino, y_treino)

def optimize_hyperparameters(X_resampled, y_resampled):
    rf_params = {
        'n_estimators': (50, 200),
        'max_depth': (2, 10),
        'max_features': ['sqrt', 'log2'],
        'min_samples_split': (2, 10)
    }

    rf_modelo = BayesSearchCV(
        RandomForestClassifier(),
        search_spaces=rf_params,
        cv=5,
        n_jobs=-1,
        verbose=1,
        random_state=42
    )
    rf_modelo.fit(X_resampled, y_resampled)
    return rf_modelo.best_params_
```

```
def train_model(X_resampled, y_resampled, best_params):
    modelo = RandomForestClassifier(**best_params, random_state=42)
    modelo.fit(X_resampled, y_resampled)
    return modelo

def evaluate_model(modelo, X_teste, y_teste):
    predictions = modelo.predict(X_teste)
    print("Acurácia:", accuracy_score(y_teste, predictions))
    print("Matriz de Confusão:\n", confusion_matrix(y_teste, predictions))
    print("Relatório de Classificação:\n", classification_report(y_teste, predictions))

training_data, test_data = load_data()
X_treino, y_treino, X_teste, y_teste = preprocess_data(training_data, test_data)
X_resampled, y_resampled = balance_data(X_treino, y_treino)
best_params = optimize_hyperparameters(X_resampled, y_resampled)
model = train_model(X_resampled, y_resampled, best_params)
evaluate_model(model, X_teste, y_teste)
```

Questao 3

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import KNNImputer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from yellowbrick.classifier import ConfusionMatrix
from imblearn.over_sampling import SMOTE
from skopt import BayesSearchCV
from sklearn import tree

def load_data():
    training_data = pd.read_csv('titanic/train.csv')
    test_data = pd.read_csv('titanic/test.csv')
    truth_table = pd.read_csv('titanic/gender_submission.csv')
    test_data = test_data.merge(truth_table, on='PassengerId', how='left')
    return training_data, test_data

def preprocess_data(training_data, test_data):
    columns_to_drop = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'Embarked']
    encoder = LabelEncoder()
    training_data['Sex'] = encoder.fit_transform(training_data['Sex'])
    test_data['Sex'] = encoder.transform(test_data['Sex'])

    imputer = KNNImputer( n_neighbors=5 )
    columns_to_input = ['Age', 'Fare']
    training_data[columns_to_input] = imputer.fit_transform( training_data[columns_to_input] )
    test_data[columns_to_input] = imputer.transform( test_data[columns_to_input] )

    X_treino = training_data.drop(columns=columns_to_drop + ['Survived'], axis=1)
    y_treino = training_data['Survived']
    X_teste = test_data.drop(columns=columns_to_drop + ['Survived'], axis=1)
    y_teste = test_data['Survived']

    return X_treino, y_treino, X_teste, y_teste

def balance_data(X_treino, y_treino):
    smote = SMOTE(random_state=42)
    return smote.fit_resample(X_treino, y_treino)

def optimize_hyperparameters(X_resampled, y_resampled):
    rf_params = {
        'n_estimators': (50, 200),
        'max_depth': (2, 10),
        'max_features': ['sqrt', 'log2'],
        'min_samples_split': (2, 10)
    }

    rf_modelo = BayesSearchCV(
        RandomForestClassifier(),
        search_spaces=rf_params,
        cv=5,
        n_jobs=-1,
        verbose=1,
        random_state=42
    )
    rf_modelo.fit(X_resampled, y_resampled)
    return rf_modelo.best_params_
```

```
def train_model(X_resampled, y_resampled, best_params):
    modelo = RandomForestClassifier(**best_params, random_state=42)
    modelo.fit(X_resampled, y_resampled)
    return modelo

def evaluate_model(modelo, X_teste, y_teste):
    predictions = modelo.predict(X_teste)
    print("Acurácia:", accuracy_score(y_teste, predictions))
    print("Matriz de Confusão:\n", confusion_matrix(y_teste, predictions))
    print("Relatório de Classificação:\n", classification_report(y_teste, predictions))

training_data, test_data = load_data()
X_treino, y_treino, X_teste, y_teste = preprocess_data(training_data, test_data)
X_resampled, y_resampled = balance_data(X_treino, y_treino)
best_params = optimize_hyperparameters(X_resampled, y_resampled)
model = train_model(X_resampled, y_resampled, best_params)
evaluate_model(model, X_teste, y_teste)
```