KNN Python Example

write python code to create a machine learning example of KNN

Sure! Below is a simple example of how to implement a K-Nearest Neighbors (KNN) algorithm using Python and the popular machine learning library scikit-learn. We'll use the Iris dataset, which is a classic dataset for classification problems.

Step-by-Step Implementation of KNN using Scikit-learn

1. **Import necessary libraries**: We'll import the necessary libraries for data manipulation, visualization, and machine learning.
2. **Load the dataset**: We'll load the Iris dataset.
3. **Preprocess the data**: We'll split the data into training and testing sets.
4. **Train the model**: We'll train the KNN model on the training data.
5. **Evaluate the model**: We'll evaluate the model's performance on the test data.

Here's the complete code:

```python
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=3)

# Train the model
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print the classification report
print('Classification Report:')
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Additional code for visualization (optional)
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Reduce dimensions to 2 for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot the decision boundary
h = .02  # step size in the mesh
x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# Fit KNN with reduced dimensions
knn_pca = KNeighborsClassifier(n_neighbors=3)
knn_pca.fit(X_pca, y)

Z = knn_pca.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, edgecolor='k', s=20, cmap=plt.cm.Paired)
plt.title('KNN Decision Boundary with Iris Data (PCA-reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

Explanation of the Code

1. **Imports**:
    - `numpy`, `pandas`: For data manipulation.
    - `scikit-learn` modules: For machine learning tasks (datasets, model selection, preprocessing, KNN, and evaluation).
    - `matplotlib`: For optional visualization.
2. **Loading the Iris dataset**:
    - The `load_iris` function loads the dataset, which contains features and target variables.
3. **Data Splitting**:
    - `train_test_split` is used to split the data into training and testing sets.
4. **Data Standardization**:
    - `StandardScaler` is used to standardize the features for better performance of the KNN algorithm.
5. **Model Training**:

- `KNeighborsClassifier` is initialized with `n_neighbors=3` (choosing 3 nearest neighbors).
- The model is trained using the `fit` method on the training data.

6. **Model Prediction and Evaluation**:

    - Predictions are made on the test set using the `predict` method.
    - `accuracy_score` and `classification_report` are used to evaluate the model's performance.

7. **Visualization (Optional)**:

    - PCA (Principal Component Analysis) is used to reduce the dimensions of the data for visualization.
    - A decision boundary plot is created to visualize the performance of the KNN classifier.

This code provides a comprehensive overview of implementing KNN using scikit-learn, including data preparation, model training, evaluation, and optional visualization.

remove the dimensionality reduction and add a search to find the best model

Sure! Let's modify the code to remove the dimensionality reduction and add a hyperparameter search to find the best KNN model. We'll use `GridSearchCV` from scikit-learn to perform the hyperparameter tuning.

Here's the updated code:

```python
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Set up the parameter grid for hyperparameter tuning
param_grid = {
    'n_neighbors': range(1, 31),
    'weights': ['uniform', 'distance'],
```

```
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Initialize GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, scoring='accuracy',
n_jobs=-1)

# Perform the grid search on the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters found: ", grid_search.best_params_)

# Train the best model found on the full training data
best_knn = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print the classification report
print('Classification Report:')
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Additional code for visualization (optional)
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Plot the decision boundary
h = .02  # step size in the mesh
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# Fit KNN with training data
best_knn.fit(X_train[:, :2], y_train)  # Use only the first two features for visualization
Z = best_knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolor='k', s=20,
cmap=plt.cm.Paired)
plt.title('KNN Decision Boundary with Iris Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Explanation of the Added/Modified Parts

1. **Parameter Grid for Hyperparameter Tuning**:

    - **`param_grid`**: A dictionary specifying the parameters and the values to be tested. Here, we are testing different values for **`n_neighbors`**, **`weights`**, and **`metric`**.

2. **GridSearchCV**:

    - **`GridSearchCV`**: A function that performs an exhaustive search over the specified parameter grid. It uses cross-validation (**`cv=5`** in this case) to evaluate each combination of parameters.

3. **Finding the Best Hyperparameters**:

    - `grid_search.fit(X_train, y_train)`: This fits the grid search model on the training data.
    - `grid_search.best_params_`: This prints the best hyperparameters found by the grid search.
    - `grid_search.best_estimator_`: This retrieves the best KNN model based on the grid search.

4. **Training and Evaluating the Best Model**:

    - The best model is then trained on the training data, predictions are made on the test data, and the model's performance is evaluated.

5. **Visualization**:

    - The visualization part remains optional. It now uses only the first two features of the training data for plotting the decision boundary.

This code provides a comprehensive overview of implementing KNN using scikit-learn, including hyperparameter tuning, model training, evaluation, and optional visualization.

give me a function that explain the prediction of a knn model

Certainly! To explain the prediction of a KNN model, we can create a function that provides insights into how the prediction was made. Specifically, we can look at the nearest neighbors that influenced the prediction. This function will use the distances and indices of the nearest neighbors to explain the prediction.

Here's the function:

```python
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

def explain_knn_prediction(model, X_train, y_train, X_test_instance, k):
    """
    Explain the prediction of a KNN model for a single test instance.

    Parameters:
    model (KNeighborsClassifier): Trained KNN model.
    X_train (numpy.ndarray): Training data features.
    y_train (numpy.ndarray): Training data labels.
    X_test_instance (numpy.ndarray): Test instance to explain.
    k (int): Number of neighbors to consider.

    Returns:
    dict: Explanation of the prediction.
    """
    # Find the k nearest neighbors
    distances, indices = model.kneighbors([X_test_instance], n_neighbors=k)

    # Retrieve the neighbors' labels
```

```
    neighbor_labels = y_train[indices[0]]

    # Count the occurrence of each class in the neighbors
    class_counts = np.bincount(neighbor_labels)

    # Get the predicted class
    predicted_class = np.argmax(class_counts)

    explanation = {
        "Test instance": X_test_instance,
        "Neighbors' indices": indices[0],
        "Neighbors' distances": distances[0],
        "Neighbors' labels": neighbor_labels,
        "Class counts": class_counts,
        "Predicted class": predicted_class
    }

    return explanation

# Example usage:
# Load the dataset and train the model
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Explain a prediction
X_test_instance = X_test[0]  # Example test instance
explanation = explain_knn_prediction(knn, X_train, y_train, X_test_instance, k=5)

# Print the explanation
print("Explanation of KNN prediction:")
for key, value in explanation.items():
    print(f"{key}: {value}")
```

Explanation of the Function

1. **Function Parameters**:
    - `model`: The trained KNN model.
    - `X_train`: Training data features.
    - `y_train`: Training data labels.
    - `X_test_instance`: The test instance to explain.
    - `k`: The number of nearest neighbors to consider (typically the same as the number used in training).
2. **Finding Nearest Neighbors**:
    - `model.kneighbors([X_test_instance], n_neighbors=k)`: Finds the `k` nearest neighbors to the test instance. It returns the distances to the neighbors and their indices in the training set.
3. **Retrieving Neighbors' Labels**:
    - `neighbor_labels = y_train[indices[0]]`: Retrieves the labels of the nearest neighbors.
4. **Counting Class Occurrences**:

- `class_counts = np.bincount(neighbor_labels)`: Counts the occurrences of each class among the neighbors.

5. **Predicted Class**:

   - `predicted_class = np.argmax(class_counts)`: Determines the class with the highest count (the predicted class).

6. **Explanation Dictionary**:

   - An explanation dictionary is created containing the test instance, neighbors' indices and distances, neighbors' labels, class counts, and the predicted class.

Example Usage

The example usage section shows how to load the Iris dataset, train the KNN model, and explain the prediction for a specific test instance.

By running this code, you will get a detailed explanation of the KNN prediction, including the nearest neighbors' indices, their distances to the test instance, their labels, the count of each class among the neighbors, and the predicted class.

give me a param grid to grid search in KNN

Certainly! Below is an example of a parameter grid (`param_grid`) that you can use to perform a grid search for hyperparameter tuning in a KNN model using `GridSearchCV` from scikit-learn. This grid explores a variety of values for the number of neighbors (`n_neighbors`), the weighting of neighbors (`weights`), and the distance metrics (`metric`).

```python
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```python
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_neighbors': range(1, 31),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Initialize GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, scoring='accuracy',
n_jobs=-1)

# Perform the grid search on the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters found: ", grid_search.best_params_)

# Train the best model found on the full training data
best_knn = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print the classification report
print('Classification Report:')
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

Explanation of the Parameter Grid

1. `n_neighbors`: This range explores the number of neighbors to use, from 1 to 30.
    - `range(1, 31)`: Tests values from 1 to 30 inclusive.
2. `weights`: Determines the weight function used in prediction.
    - `'uniform'`: All neighbors have equal weight.
    - `'distance'`: Closer neighbors have a higher influence.
3. `metric`: The distance metric to use for the tree.
    - `'euclidean'`: Standard Euclidean distance.
    - `'manhattan'`: Manhattan distance (L1 norm).
    - `'minkowski'`: Generalized distance metric (Euclidean and Manhattan are special cases of Minkowski).

Steps to Perform Grid Search

1. **Data Preparation**:
    - Load and split the Iris dataset.
    - Standardize the features.
2. **Define the Parameter Grid**:
    - Specify the values for `n_neighbors`, `weights`, and `metric`.
3. **Initialize and Perform Grid Search**:

    - Initialize the `KNeighborsClassifier`.
    - Use `GridSearchCV` to search for the best combination of hyperparameters with 5-fold cross-validation (`cv=5`).
    - Fit the grid search on the training data.

4. **Evaluate the Best Model**:

    - Retrieve and print the best hyperparameters.
    - Train the best model on the full training data.
    - Make predictions on the test set and evaluate the model's performance.

This code snippet performs hyperparameter tuning for a KNN model, helping to find the optimal configuration for the given dataset.

Recomendações para quem tem suspeita de câncer de mama

Se você ou alguém que conhece tem suspeita de câncer de mama, é essencial tomar medidas apropriadas rapidamente. A seguir, algumas recomendações importantes:

1. Consulte um Profissional de Saúde

- **Agende uma consulta**: Marque uma consulta com um médico, de preferência um especialista em oncologia ou um mastologista, que é especializado em doenças da mama.
- **Descreva os sintomas**: Explique detalhadamente quaisquer sintomas que esteja experimentando, como caroços na mama, alterações no formato ou tamanho, alterações na pele, dor ou secreção do mamilo.

2. Realize Exames de Diagnóstico

- **Mamografia**: Este é um exame de imagem da mama que pode ajudar a detectar alterações suspeitas.
- **Ultrassonografia**: Utilizada para complementar a mamografia, especialmente em mamas densas.
- **Ressonância Magnética**: Pode ser recomendada para casos mais complexos ou para uma avaliação mais detalhada.
- **Biópsia**: Se um nódulo ou outra alteração suspeita for encontrada, pode ser necessária uma biópsia para determinar se é cancerígeno. Existem diferentes tipos de biópsias, como a biópsia por agulha fina, biópsia por agulha grossa e biópsia cirúrgica.

3. Histórico Familiar e Genética

- **Converse sobre seu histórico familiar**: Informe seu médico sobre qualquer histórico familiar de câncer de mama ou outros tipos de câncer.
- **Teste Genético**: Se houver histórico familiar significativo, seu médico pode recomendar testes genéticos para verificar mutações nos genes BRCA1 e BRCA2, que aumentam o risco de câncer de mama.

4. Informação e Apoio Psicológico

- **Eduque-se**: Aprender mais sobre o câncer de mama pode ajudá-lo a entender o diagnóstico e as opções de tratamento.
- **Apoio emocional**: Considere procurar apoio psicológico ou grupos de apoio. O diagnóstico ou a suspeita de câncer pode ser emocionalmente desafiador, e falar com outros que passaram por experiências similares pode ser reconfortante.

5. Estilo de Vida Saudável

- **Alimentação saudável**: Mantenha uma dieta equilibrada rica em frutas, legumes e grãos integrais.
- **Exercício físico**: A prática regular de atividade física pode ajudar a melhorar a saúde geral e o bem-estar.
- **Evite álcool e tabaco**: Reduzir ou eliminar o consumo de álcool e evitar o tabaco pode diminuir o risco de câncer e melhorar a saúde geral.

6. Monitoramento Contínuo

- **Exames regulares**: Mesmo que a suspeita não se confirme, continue fazendo exames de rastreamento conforme recomendado pelo seu médico, especialmente se você tiver fatores de risco elevados.
- **Autoexame**: Aprenda a fazer o autoexame da mama e realize-o regularmente, mas lembre-se de que ele não substitui exames médicos profissionais.

7. Planejamento do Tratamento

- **Discuta as opções de tratamento**: Se o câncer de mama for diagnosticado, converse detalhadamente sobre as opções de tratamento com seu médico. Isso pode incluir cirurgia, radioterapia, quimioterapia, terapia hormonal ou terapias direcionadas.
- **Segunda opinião**: Não hesite em buscar uma segunda opinião médica para se sentir mais seguro sobre o diagnóstico e o plano de tratamento.

Lembre-se de que a detecção precoce e o tratamento adequado são cruciais para um melhor prognóstico no caso de câncer de mama. Siga as recomendações do seu médico e mantenha um diálogo aberto sobre suas preocupações e opções.

Em um resumo de um parágrafo, dê recomendações para quem tem suspeita de câncer de mama positivo

Se você recebeu um diagnóstico positivo de câncer de mama, é crucial seguir algumas recomendações importantes: agende consultas com um oncologista ou mastologista para discutir seu diagnóstico e plano de tratamento, que pode incluir cirurgia, quimioterapia, radioterapia e/ou terapia hormonal. Realize todos os exames necessários, como mamografia, ultrassonografia e ressonância magnética, para avaliar a extensão do câncer. Informe-se sobre o seu tipo específico de câncer e suas opções terapêuticas, buscando também uma segunda opinião médica se desejar. Além disso, procure apoio emocional, seja através de grupos de apoio ou aconselhamento psicológico, para lidar com o impacto emocional do diagnóstico. Adote um estilo de vida saudável, mantendo uma alimentação equilibrada e realizando atividades físicas regularmente, e evite álcool e tabaco. Monitorize regularmente seu progresso com seu médico e siga todas as orientações de tratamento e acompanhamento.

Em um resumo de um parágrafo, dê recomendações para quem não tem suspeita de câncer de mama e deseja se prevenir

Para quem não tem suspeita de câncer de mama e deseja se prevenir, é essencial adotar um estilo de vida saudável e realizar exames de rastreamento regulares. Mantenha uma dieta equilibrada rica em frutas, vegetais e grãos integrais, pratique atividade física regularmente, evite o consumo excessivo de álcool e não fume. Realize mamografias periódicas conforme a orientação médica, especialmente se você tiver mais de 40 anos ou histórico familiar de câncer de mama. Aprenda a fazer o autoexame das mamas mensalmente para conhecer o aspecto normal do seu corpo e identificar qualquer alteração rapidamente. Considere também discutir com seu médico a possibilidade de testes genéticos se houver histórico familiar significativo. A detecção precoce e um estilo de vida saudável são fundamentais para reduzir o risco de câncer de mama.