Aula 26 - Assícrona - Atividade Prática 4

Implementação de modelos e verificação de eficiência da estratégia adotada com redes neurais e busca de parâmetros por grid search/random search

Grupo 15

Nome: Ubiratan da Silva Tavares - RA: 23031559

Otimização de hiperparâmetros do modelo

- Os modelos de aprendizado de máquina têm hiperparâmetros que podem ser definidos para configurar um modelo de acordo com a base de dados de treinamento.
- A melhor forma para se definir um hiperparâmetro e combinações de hiperparâmetros para um determinado conjunto de dados é um desafio.
- A melhor abordagem é realizar pesquisar com diferentes valores para os hiperparâmetros do modelo e escolher um subconjunto que resulte em um modelo que alcance o melhor desempenho em um determinado conjunto de dados (otimização de hiperparâmetros ou ajuste de hiperparâmetro)
- O resultado de uma otimização de hiperparâmetros é um conjunto único de hiperparâmetros de bom desempenho que pode ser usado para configurar o modelo.
- Um ponto no espaço de busca é um vetor com um valor específico para cada valor de hiperparâmetro.
- O objetivo do procedimento de otimização é encontrar um vetor que resulte no melhor desempenho do modelo após o aprendizado, como precisão máxima ou erro mínimo.
- Uma variedade de algoritmos de otimização diferentes podem ser usados, entretanto os métodos de busca em grade e busca aleatória são os mais comuns.
    - **Pesquisa em grade**: define um espaço de pesquisa como uma grade de valores de hiperparâmetros e avalie cada posição na grade. É um método ideal para combinações de verificação pontual que geralmente apresentam bom desempenho.
    - **Pesquisa aleatória**: define um espaço de pesquisa como um domínio limitado de valores de hiperparâmetros e amostra aleatoriamente pontos nesse domínio. É um método ideal descobrir e obter combinações de hiperparâmetros que não foram considerados na pesquisa em grade.

API Scikit-Learn de otimização de hiperparâmetros

- A biblioteca scikit-learn fornece técnicas para ajustar hiperparâmetros de modelo.
    - A classe GridSearchCV para pesquisa em grade.
    - A classe RandomizedSearchCV para pesquisa aleatória.
- Ambas as técnicas acima avaliam modelos para um determinado vetor de hiperparâmetros usando validação cruzada.
- Ambas as classes requerem dois argumentos:
    - O primeiro é o modelo a ser otimizado.
    - O segundo é o espaço de busca.

In [1]:
```
# importando as bibliotecas
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train_test_split
from sklearn.metrics import classification_report
```

In [2]:
```
# importando a base dados
X, y = load_breast_cancer(return_X_y=True)
print(X.shape, y.shape)
```

```
(569, 30) (569,)
```

In [3]:
```
# dividindo a base de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [4]:
```
# criando o modelo classificador
mlp = MLPClassifier(max_iter=1000, random_state=42)
```

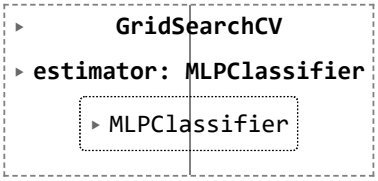
Pesquisa em grade para classificação

In [5]:
```
# definindo o espaço de busca para os hiperparâmetros
space = {'hidden_layer_sizes': [(64, 64), (128, 128)],
         'activation': ['tanh', 'relu'],
         'solver': ['sgd', 'adam'],
         'alpha': [0.0001, 0.001],
         'learning_rate': ['constant', 'adaptive']}
```

In [6]:
```
# criando o objeto da classe GridSearchCV
grid_search = GridSearchCV(estimator=mlp, param_grid=space, cv=5, n_jobs=1)
```

In [7]:
```
# realizando a busca em grade com a Validação Cruzada
grid_search.fit(X_train, y_train)
```

Out[7]:



In [8]:
```
# obtendo os melhores hiperparâmetros encontrados na pesquisa em grade
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"A melhor performance do modelo foi {best_score:.2f}")
print(f"Os melhores hiperparâmatros encontrados: {best_params}")
```

```
A melhor performance do modelo foi 0.93
Os melhores hiperparâmatros encontrados: {'activation': 'relu', 'alpha': 0.0001, 'hi
dden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'adam'}
```

In [9]:

```
medias = grid_search.cv_results_['mean_test_score']
desvios = grid_search.cv_results_['std_test_score']
parametros = grid_search.cv_results_['params']

# combinando as médias, desvios e parâmetros em uma lista de tuplas
resultados = list(zip(medias, desvios, parametros))

# classificando a lista de tuplas com base na média em ordem decrescente
resultados.sort(reverse=True, key=lambda x: x[0])

contador = 1
# imprimindo os resultados em ordem decrescente
for media, desvio, parametro in resultados:
    print(f"{contador}: {media:.2f} ({desvio:.2f}): {parametro}")
    contador += 1
```

1: 0.93 (0.03): {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'adam'}
2: 0.93 (0.03): {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'adaptive', 'solver': 'adam'}
3: 0.92 (0.04): {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'constant', 'solver': 'adam'}
4: 0.92 (0.04): {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'adam'}
5: 0.92 (0.05): {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'adam'}
6: 0.92 (0.05): {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'adaptive', 'solver': 'adam'}
7: 0.92 (0.03): {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'adam'}
8: 0.92 (0.03): {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'adaptive', 'solver': 'adam'}
9: 0.92 (0.05): {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'constant', 'solver': 'adam'}
10: 0.92 (0.05): {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'adam'}
11: 0.92 (0.04): {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'adam'}
12: 0.92 (0.04): {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'adaptive', 'solver': 'adam'}
13: 0.91 (0.04): {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'constant', 'solver': 'adam'}
14: 0.91 (0.04): {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'adam'}
15: 0.91 (0.04): {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'adaptive', 'solver': 'sgd'}
16: 0.91 (0.05): {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'constant', 'solver': 'adam'}
17: 0.91 (0.05): {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'adam'}
18: 0.90 (0.05): {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'sgd'}
19: 0.89 (0.05): {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'constant', 'solver': 'sgd'}
20: 0.89 (0.05): {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'adaptive', 'solver': 'sgd'}
21: 0.89 (0.04): {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'constant', 'solver': 'sgd'}
22: 0.89 (0.04): {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'sgd'}
23: 0.89 (0.04): {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'sgd'}
24: 0.87 (0.05): {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'sgd'}
25: 0.68 (0.11): {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'sgd'}
26: 0.68 (0.10): {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'adaptive', 'solver': 'sgd'}
27: 0.68 (0.11): {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'constant', 'solver': 'sgd'}
28: 0.68 (0.10): {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (128, 128), 'learning_rate': 'adaptive', 'solver': 'sgd'}
29: 0.67 (0.26): {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'sgd'}
30: 0.67 (0.26): {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'sgd'}
31: 0.65 (0.26): {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'constant', 'solver': 'sgd'}
32: 0.64 (0.24): {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'constant', 'solver': 'sgd'}

In [10]:
```
# treinando o modelo com os melhores hiperparâmetros com os dados de treinamento
best_mlp = MLPClassifier(max_iter=1000, random_state=42, **best_params)
```

In [11]:
```
# ajustando os dados de treinamento npara treinamento do modelo classificador
best_mlp.fit(X_train, y_train)
```

Out[11]:
```
MLPClassifier(hidden_layer_sizes=(128, 128), max_iter=1000, random_state=42)
```

In [12]:
```
# fazendo as previsões do modelo treinado com os dados de teste
y_pred = best_mlp.predict(X_test)
```

In [13]:
```
# avaliando o desempenho do modelo classificador
report = classification_report(y_test, y_pred)
print("\nRelatório de Classificação:")
print(report)
```

```
Relatório de Classificação:
              precision    recall  f1-score   support

           0       0.81      0.98      0.88        43
           1       0.98      0.86      0.92        71

    accuracy                           0.90       114
   macro avg       0.90      0.92      0.90       114
weighted avg       0.92      0.90      0.90       114
```

Análise do Desempenho do Modelo Classificador com uso dos melhores hiperparâmetros obtidos com a técnica da Pesquisa em Grade

- **Precision (Precisão):** Para a classe 0, a precisão é de 0.81, o que significa que 81% das previsões da classe 0 estão corretas. Para a classe 1, a precisão é de 0.98, o que indica que 98% das previsões da classe 1 estão corretas.
- **Recall (Revocação):** Para a classe 0, o recall é de 0.98, o que significa que o modelo identificou 98% dos verdadeiros exemplos da classe 0. Para a classe 1, o recall é de 0.86, indicando que o modelo identificou 86% dos verdadeiros exemplos da classe 1.
- **F1-score:** O F1-score é uma métrica que combina precisão e recall em uma única medida. Para a classe 0, o F1-score é 0.88, e para a classe 1, é 0.92.
- **Accuracy (Precisão Global):** A precisão global do modelo é de 90%. Isso significa que o modelo classificou corretamente 90% das amostras no conjunto de dados.

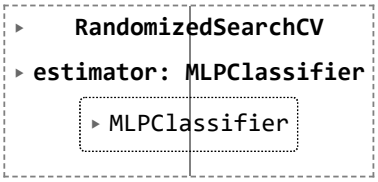
Pesquisa em aleatória para classificação

In [24]:
```
# definindo o espaço de busca para os hiperparâmetros
space = {'hidden_layer_sizes': [(64, 64), (128, 128)],
         'activation': ['tanh', 'relu'],
         'solver': ['sgd', 'adam'],
         'alpha': np.logspace(-4, 0, 5),
         'learning_rate': ['constant', 'adaptive']}
```

In [25]:
```
# criando o objeto da classe RandomizedSearchCV
random_search = RandomizedSearchCV(mlp, param_distributions=space, n_iter=10, cv=5, n_
```

In [26]:
```
# realizando a busca aleatória com a Validação Cruzada
random_search.fit(X_train, y_train)
```

Out[26]:



In [27]:
```
# obtendo os melhores hiperparâmetros encontrados na pesquisa em grade
best_params = random_search.best_params_
best_score = random_search.best_score_
print(f"A melhor performance do modelo foi {best_score:.2f}")
print(f"Os melhores hiperparâmatros encontrados: {best_params}")
```

```
A melhor performance do modelo foi 0.93
Os melhores hiperparâmatros encontrados: {'solver': 'adam', 'learning_rate': 'adapti
ve', 'hidden_layer_sizes': (128, 128), 'alpha': 0.1, 'activation': 'tanh'}
```

In [28]:
```
medias = random_search.cv_results_['mean_test_score']
desvios = random_search.cv_results_['std_test_score']
parametros = random_search.cv_results_['params']

# combinando as médias, desvios e parâmetros em uma lista de tuplas
resultados = list(zip(medias, desvios, parametros))

# classificando a lista de tuplas com base na média em ordem decrescente
resultados.sort(reverse=True, key=lambda x: x[0])

contador = 1
# imprimindo os resultados em ordem decrescente
for media, desvio, parametro in resultados:
    print(f"{contador}: {media:.2f} ({desvio:.2f}): {parametro}")
    contador += 1
```

```
1: 0.93 (0.03): {'solver': 'adam', 'learning_rate': 'adaptive', 'hidden_layer_size
s': (128, 128), 'alpha': 0.1, 'activation': 'tanh'}
2: 0.90 (0.05): {'solver': 'sgd', 'learning_rate': 'adaptive', 'hidden_layer_sizes':
(64, 64), 'alpha': 0.001, 'activation': 'tanh'}
3: 0.89 (0.04): {'solver': 'sgd', 'learning_rate': 'adaptive', 'hidden_layer_sizes':
(128, 128), 'alpha': 0.01, 'activation': 'tanh'}
4: 0.89 (0.05): {'solver': 'sgd', 'learning_rate': 'adaptive', 'hidden_layer_sizes':
(128, 128), 'alpha': 0.1, 'activation': 'tanh'}
5: 0.89 (0.04): {'solver': 'sgd', 'learning_rate': 'adaptive', 'hidden_layer_sizes':
(64, 64), 'alpha': 0.01, 'activation': 'tanh'}
6: 0.89 (0.04): {'solver': 'sgd', 'learning_rate': 'constant', 'hidden_layer_sizes':
(64, 64), 'alpha': 0.0001, 'activation': 'tanh'}
7: 0.89 (0.04): {'solver': 'sgd', 'learning_rate': 'constant', 'hidden_layer_sizes':
(128, 128), 'alpha': 0.0001, 'activation': 'tanh'}
8: 0.87 (0.05): {'solver': 'sgd', 'learning_rate': 'constant', 'hidden_layer_sizes':
(128, 128), 'alpha': 0.001, 'activation': 'tanh'}
9: 0.86 (0.05): {'solver': 'sgd', 'learning_rate': 'constant', 'hidden_layer_sizes':
(128, 128), 'alpha': 0.1, 'activation': 'tanh'}
10: 0.69 (0.12): {'solver': 'sgd', 'learning_rate': 'adaptive', 'hidden_layer_size
s': (128, 128), 'alpha': 0.1, 'activation': 'relu'}
```

In [29]:
```
# treinando o modelo com os melhores hiperparâmetros com os dados de treinamento
best_mlp = MLPClassifier(max_iter=1000, random_state=42, **best_params)
```

In [30]:
```
# ajustando os dados de treinamento npara treinamento do modelo classificador
best_mlp.fit(X_train, y_train)
```

Out[30]:
```
MLPClassifier(activation='tanh', alpha=0.1, hidden_layer_sizes=(128, 128),
              learning_rate='adaptive', max_iter=1000, random_state=42)
```

In [31]:
```
# fazendo as previsões do modelo treinado com os dados de teste
y_pred = best_mlp.predict(X_test)
```

In [32]:
```
# avaliando o desempenho do modelo classificador
report = classification_report(y_test, y_pred)
print("\nRelatório de Classificação:")
print(report)
```

```
Relatório de Classificação:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        43
           1       0.96      0.97      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Análise do Desempenho do Modelo Classificador com uso dos melhores hiperparâmetros obtidos com a técnica da Pesquisa Aleatória

- **Precision (Precisão):** Para a classe 0, a precisão é de 0.95, indicando que 95% das previsões da classe 0 estão corretas. Para a classe 1, a precisão é de 0.96, o que significa que 96% das previsões da classe 1 estão corretas.
- **Recall (Revocação):** Para a classe 0, o recall é de 0.93, indicando que o modelo identificou 93% dos verdadeiros exemplos da classe 0. Para a classe 1, o recall é de 0.97, o que significa que o modelo identificou 97% dos verdadeiros exemplos da classe 1.
- **F1-score:** O F1-score é uma métrica que combina precisão e recall em uma única medida. Para a classe 0, o F1-score é 0.94, e para a classe 1, é 0.97.
- **Accuracy (Precisão Global):** A precisão global do modelo é de 96%. Isso significa que o modelo classificou corretamente 96% das amostras no conjunto de dados.

Comparação entre os métodos de pesquisa em grade e pesquisa aleatória

- O método de pesquisa aleatória (`RandomizedSearchCV`) obteve resultados ligeiramente melhores em termos de precisão, recall e F1-score para ambas as classes (0 e 1) em comparação com o método de pesquisa em grade (`GridSearchCV`).
- Além disso, o método de pesquisa aleatória alcançou uma precisão global (accuracy) de 96%, enquanto o método de pesquisa em grade alcançou uma precisão global de 90%.
- Isso sugere que a pesquisa aleatória conseguiu encontrar uma combinação de hiperparâmetros que resultou em um modelo com melhor desempenho geral no conjunto de dados.

Portanto, com base nos resultados apresentados, o método de pesquisa aleatória parece ter sido mais eficaz na otimização dos hiperparâmetros do modelo em comparação com o método de pesquisa em grade para o seu conjunto de dados específico.