Projeto Final - EC37C - Inteligência Artificial

Considerando a base de dados Titanic: https://www.kaggle.com/datasets/yasserh/titanic-dataset

Alunos: Felipe - Neidielli Alves Rosado - Vinicius Dias

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, StratifiedKFold, cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

Carregamento e pré-processamento dos dados sem remover 'Cabin'.

```
# Carregar os dados
data = pd.read_csv('Titanic-Dataset.csv')

# Remover colunas irrelevantes para a classificação
data = data.drop(['PassengerId', 'Name', 'Ticket'], axis=1)

# Lidar com valores ausentes
data['Age'].fillna(data['Age'].mean(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Codificar variáveis categóricas
label_encoder = LabelEncoder()
data['Sex'] = label_encoder.fit_transform(data['Sex'])
data['Embarked'] = label_encoder.fit_transform(data['Embarked'])
data['Cabin'] = label_encoder.fit_transform(data['Cabin'])

# Dividir os dados em features (X) e rótulos (y)
X = data.drop('Survived', axis=1)
y = data['Survived']

# Padronizar as features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Balanceamento das Classes

```
# Balanceamento das classes usando SMOTE
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)
```

Aplicação da separação do conjunto treino (70%) e teste (30%).

```
# Divisão dos dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Redes Neurais com Grid

```
# Definir os parâmetros para busca
parameters = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50), (100,100, 50)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['adam'],
    'alpha': [0.0001, 0.001, 0.01]
}

# Criar o modelo MLP
mlp = MLPClassifier(random_state=42, max_iter=1500)
```

```
# Criar o objeto GridSearchCV
grid_search = GridSearchCV(mlp, parameters, cv=10)

# Treinar o modelo com os dados de treinamento
grid_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Treinar o modelo com os dados de treinamento
best_estimator.fit(X_train, y_train)

# Obter as previsões no conjunto de teste
y_pred = best_estimator.predict(X_test)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Calcular a acurácia usando a função accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

Redes Neurais com RandomSearch

```
# Criar o objeto RandomizedSearchCV
random_search = RandomizedSearchCV(mlp, parameters, cv=10)

# Treinar o modelo com os dados de treinamento
random_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = random_search.best_params_
best_estimator = random_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```
# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Arvore de decisão com GRID

```
# Definir os hiperparâmetros a serem ajustados
parameters = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Criar o modelo de árvore de decisão
dt = DecisionTreeClassifier(random_state=42)

# Criar o objeto GridSearchCV
grid_search = GridSearchCV(dt, parameters, cv=10)

# Treinar o modelo com os dados de treinamento
grid_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Arvore de Decisão com Random

```
# Criar o objeto RandomizedSearchCV
random_search = RandomizedSearchCV(dt, parameters, cv=10)

# Treinar o modelo com os dados de treinamento
random_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = random_search.best_params_
best_estimator = random_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Random Forest

Ajuste dos hiperparâmetros com GridSearchCV

```
# Definir os hiperparâmetros para busca
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Criar o objeto do classificador de Floresta Aleatória
rf = RandomForestClassifier(random_state=0)

# Realizar a busca de hiperparâmetros com GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=10)
grid_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
```

```
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Treinar o modelo com os dados de treinamento
best_estimator.fit(X_train, y_train)

# Obter as previsões no conjunto de teste
y_pred = best_estimator.predict(X_test)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Calcular a acurácia usando a função accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

Random Search

```
# Criar o objeto RandomizedSearchCV
random_search = RandomizedSearchCV(rf, parameters, cv=10)

# Treinar o modelo com os dados de treinamento
random_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = random_search.best_params_
best_estimator = random_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

========== KNN sem remover 'Cabin' ==========

KNN Sem remover Cabin com GridSearch

```
# Ajuste de hiperparâmetros para KNN
parameters_knn = {'n_neighbors': [3, 5, 7, 9, 11]}
knn = KNeighborsClassifier(metric='euclidean')

# Realizar a busca de hiperparâmetros com GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=10)
grid_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Treinar o modelo com os dados de treinamento
best_estimator.fit(X_train, y_train)

# Obter as previsões no conjunto de teste
y_pred = best_estimator.predict(X_test)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Calcular a acurácia usando a função accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

KNN SEM Cabin com Random

```
# Criar o objeto RandomizedSearchCV
random_search = RandomizedSearchCV(knn, parameters, cv=10)

# Treinar o modelo com os dados de treinamento
random_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = random_search.best_params_
best_estimator = random_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')
```

```
# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

========== KNN removendo 'Cabin' ==========

```
#Remover colunas irrelevantes para a classificação
data = data.drop(['Cabin'], axis=1)


# Dividir os dados em features (X) e rótulos (y)
X = data.drop('Survived', axis=1)
#y = data['Survived'] // Somente removeu Cabin

# Padronizar as features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Divisão treino e teste - removendo Cabin

```
# Divisão dos dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Balanceamento das Classes - revomendo Cabin

```
X, y = smote.fit_resample(X, y)
```

KNN Com Cabin Grid

```
# Realizar a busca de hiperparâmetros com GridSearchCV
grid_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)
```

```
# Treinar o modelo com os dados de treinamento
best_estimator.fit(X_train, y_train)

# Obter as previsões no conjunto de teste
y_pred = best_estimator.predict(X_test)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Calcular a acurácia usando a função accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

KNN Com Cabin Random

```
# Criar o objeto RandomizedSearchCV
random_search = RandomizedSearchCV(knn, parameters, cv=10)

# Treinar o modelo com os dados de treinamento
random_search.fit(X_train, y_train)

# Obter os melhores parâmetros e o melhor estimador
best_params = random_search.best_params_
best_estimator = random_search.best_estimator_

# Avaliar o desempenho do modelo no conjunto de teste
y_pred = best_estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do modelo no conjunto de teste:", accuracy)

# Definir o objeto de amostragem por validação cruzada estratificada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Calcular a acurácia nos conjuntos de treinamento e teste usando validação cruzada estratificada
train_accuracies = cross_val_score(best_estimator, X_train, y_train, cv=cv, scoring='accuracy')
test_accuracies = cross_val_score(best_estimator, X_test, y_test, cv=cv, scoring='accuracy')

# Calcular a média e o desvio padrão da acurácia nos conjuntos de treinamento e teste
train_accuracy_mean = train_accuracies.mean()
train_accuracy_std = train_accuracies.std()
test_accuracy_mean = test_accuracies.mean()
test_accuracy_std = test_accuracies.std()

# Imprimir a acurácia nos conjuntos de treinamento e teste
print("Acurácia média nos conjuntos de treinamento:", train_accuracy_mean)
print("Acurácia média nos conjuntos de teste:", test_accuracy_mean)

# Calcular a matriz de confusão
confusion_mat = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confusão usando um mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Em execução (46s) <cel… › f › _ru… › evaluate… › _… › _… › dispatc… › _di… › appl… › _… › _… › <li… › _… › _fit_a… › f › _ › _fit_s… › _b… › _forw… › inplace…