Classification using sklearn and keras (with pandas)

File access required: In Colab this notebook requires first uploading files **Cities.csv**, **Players.csv**, and **Titanic.csv** using the *Files* feature in the left toolbar. If running the notebook on a local computer, simply ensure these files are in the same workspace as the notebook.

```
# Set-up
import csv
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from keras import Sequential
from keras.layers import Dense
from numpy.random import seed
import tensorflow
```

Prepare Cities data for classification

Predict *temperature category* from other features

```
# Read Cities.csv into dataframe, add column for temperature category
# Note: For a dataframe D and integer i, D.loc[i] is the i-th row of D
f = open('Cities.csv')
cities = pd.read_csv(f)
categories = []
for i in range(len(cities)):
    if cities.loc[i]['temperature'] < 5:
        categories.append('cold')
    elif cities.loc[i]['temperature'] < 9:
        categories.append('cool')
    elif cities.loc[i]['temperature'] < 15:
        categories.append('warm')
    else: categories.append('hot')
cities['category'] = categories
print("cold:", len(cities[(cities.category == 'cold')]))
print("cool:", len(cities[(cities.category == 'cool')]))
print("warm:", len(cities[(cities.category == 'warm')]))
print("hot:", len(cities[(cities.category == 'hot')]))
```

```
cold: 17
cool: 92
warm: 79
hot: 25
```

```
# Create training and test sets for cities data
numitems = len(cities)
percenttrain = 0.85
numtrain = int(numitems*percenttrain)

print('Training set', numtrain, 'items')
print('Test set', numitems - numtrain, 'items')

citiesTrain = cities[0:numtrain]
citiesTest = cities[numtrain:]
```

```
Training set 181 items
Test set 32 items
```

K-nearest-neighbors classification

```
features = ['longitude', 'latitude']
neighbors = 3
predict = 'category'

classifier = KNeighborsClassifier(neighbors)
classifier.fit(citiesTrain[features], citiesTrain[predict])


predictions = classifier.predict(citiesTest[features])

# Calculate accuracy
actuals = list(citiesTest[predict])
correct = 0

for i in range(len(actuals)):
  print('Predicted:', predictions[i], ' Actual:', actuals[i])
  if predictions[i] == actuals[i]: correct +=1
print('Accuracy:', round(correct/len(actuals),5))
# Comment out print, try different values for neighbors, different features
```

```
Predicted: warm  Actual: cool
Predicted: warm  Actual: warm
Predicted: hot  Actual: warm
Predicted: warm  Actual: warm
Predicted: cold  Actual: cool
Predicted: cool  Actual: cool
Predicted: cool  Actual: cool
Predicted: warm  Actual: warm
```

```
Predicted: warm  Actual: warm
Predicted: cool  Actual: cold
Predicted: cold  Actual: cold
Predicted: cool  Actual: warm
Predicted: cool  Actual: cold
Predicted: warm  Actual: warm
Predicted: warm  Actual: warm
Predicted: cool  Actual: warm
Predicted: warm  Actual: warm
Predicted: hot  Actual: hot
Predicted: cold  Actual: cold
Predicted: cold  Actual: cold
Predicted: cool  Actual: cold
Predicted: hot  Actual: hot
Predicted: warm  Actual: cool
Predicted: warm  Actual: warm
Predicted: cool  Actual: cool
Predicted: cool  Actual: cool
Predicted: cool  Actual: cool
Predicted: cool  Actual: warm
Predicted: warm  Actual: warm
Predicted: cool  Actual: cool
Predicted: warm  Actual: warm
Predicted: cool  Actual: cool
Accuracy: 0.6875
```

Your Turn: K-nearest-neighbors on World Cup data

Predict *position* from one or more of *minutes, shots, passes, tackles, saves*

```python
# This cell does all the set-up, including reordering the data to avoid team bias.
f = open('Players.csv')
players = pd.read_csv(f)
players = players.sort_values(by='surname')
players = players.reset_index(drop=True)
numitems = len(players)
percenttrain = 0.92
numtrain = int(numitems*percenttrain)
print('Training set', numtrain, 'items')
print('Test set', numitems - numtrain, 'items')
playersTrain = players[0:numtrain]
playersTest = players[numtrain:]
```

```
Training set 547 items
Test set 48 items
```

```python
# This cell does the classification.
# Try different features and different numbers of neighbors.
# What's the highest accuracy you can get?
features = ['minutes', 'shots', 'passes', 'tackles', 'saves']
neighbors = 10
predict = 'position'

classifier = KNeighborsClassifier(neighbors)
classifier.fit(playersTrain[features], playersTrain[predict])
predictions = classifier.predict(playersTest[features])

# Calculate accuracy
actuals = list(playersTest[predict])
correct = 0
for i in range(len(actuals)):
  #print('Predicted:', predictions[i], ' Actual:', actuals[i])
  if predictions[i] == actuals[i]: correct +=1
#print('Accuracy:', round(correct/len(actuals),5))
# Comment out print, try different values for neighbors, different features

print("Testing Different Configurations")
for n in [3, 5, 7, 10, 15, 20]:
    classifier = KNeighborsClassifier(n)
    classifier.fit(playersTrain[features], playersTrain[predict])
    predictions = classifier.predict(playersTest[features])
    actuals = list(playersTest[predict])
    correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
    print(f'Neighbors={n}: Accuracy = {round(correct/len(actuals), 5)}')

#Test different feature combinations
print("\nTesting different feature combinations")
feature_combos = [
    ['minutes', 'passes'],
    ['minutes', 'shots', 'passes'],
    ['minutes', 'shots', 'passes', 'tackles'],
    ['minutes', 'shots', 'passes', 'tackles', 'saves'],
]

for features in feature_combos:
    classifier = KNeighborsClassifier(10)
    classifier.fit(playersTrain[features], playersTrain[predict])
    predictions = classifier.predict(playersTest[features])
    actuals = list(playersTest[predict])
    correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
    print(f'{len(features)} features: Accuracy = {round(correct/len(actuals), 5)}')
```

```
Testing Different Configurations
Neighbors=3: Accuracy = 0.54167
Neighbors=5: Accuracy = 0.58333
Neighbors=7: Accuracy = 0.54167
Neighbors=10: Accuracy = 0.54167
Neighbors=15: Accuracy = 0.54167
Neighbors=20: Accuracy = 0.5
```

```
Testing different feature combinations
2 features: Accuracy = 0.47917
3 features: Accuracy = 0.47917
4 features: Accuracy = 0.52083
5 features: Accuracy = 0.54167
```

Your Turn Extra: K-nearest-neighbors on Titanic data - Graded

Predict *survived* from one or more of *gender, age, class, fare, embarked*

```
# This cell does all the set-up
f = open('Titanic.csv')
titanic = pd.read_csv(f)
# Convert gender and embarked to numeric values and missing ages to average age
titanic['gender'].replace({'M':0, 'F':1}, inplace=True)
titanic['embarked'].replace({'Cherbourg':0, 'Southampton':1, 'Queenstown':2}, inplace=True)
avg_age = np.average(titanic['age'].dropna().tolist())
titanic['age'].fillna(avg_age, inplace=True)
# Create training and test sets
numitems = len(titanic)
percenttrain = 0.92
numtrain = int(numitems*percenttrain)
print('Training set', numtrain, 'items')
print('Test set', numitems - numtrain, 'items')
titanicTrain = titanic[0:numtrain]
titanicTest = titanic[numtrain:]
```

```
Training set 819 items
Test set 72 items
/tmp/ipython-input-257007092.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  titanic['gender'].replace({'M':0, 'F':1}, inplace=True)
/tmp/ipython-input-257007092.py:5: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a futur
  titanic['gender'].replace({'M':0, 'F':1}, inplace=True)
/tmp/ipython-input-257007092.py:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  titanic['embarked'].replace({'Cherbourg':0, 'Southampton':1, 'Queenstown':2}, inplace=True)
/tmp/ipython-input-257007092.py:6: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a futur
  titanic['embarked'].replace({'Cherbourg':0, 'Southampton':1, 'Queenstown':2}, inplace=True)
/tmp/ipython-input-257007092.py:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  titanic['age'].fillna(avg_age, inplace=True)
```

```
# This cell does the classification.
# Try different features and different numbers of neighbors.
# What's the highest accuracy you can get?
features = ['gender', 'age', 'class', 'fare', 'embarked']
neighbors = 10
predict = 'survived'

classifier = KNeighborsClassifier(neighbors)
classifier.fit(titanicTrain[features], titanicTrain[predict])
predictions = classifier.predict(titanicTest[features])

# Calculate accuracy
actuals = list(titanicTest[predict])
correct = 0
for i in range(len(actuals)):
# print('Predicted:', predictions[i], ' Actual:', actuals[i])
  if predictions[i] == actuals[i]: correct +=1
print('Accuracy:', round(correct/len(actuals),5))

print("\nTesting Different Feature Combinations")
feature_sets = [
    ['gender'],
    ['gender', 'class'],
    ['gender', 'age', 'class'],
    ['gender', 'age', 'class', 'fare'],
    ['gender', 'age', 'class', 'fare', 'embarked']
]

for features in feature_sets:
    classifier = KNeighborsClassifier(10)
    classifier.fit(titanicTrain[features], titanicTrain[predict])
    predictions = classifier.predict(titanicTest[features])
    actuals = list(titanicTest[predict])
    correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
    print(f'{features}: Accuracy = {round(correct/len(actuals), 5)}')
```

```
Accuracy: 0.73611

Testing Different Feature Combinations
['gender']: Accuracy = 0.81944
['gender', 'class']: Accuracy = 0.86111
['gender', 'age', 'class']: Accuracy = 0.73611
['gender', 'age', 'class', 'fare']: Accuracy = 0.75
['gender', 'age', 'class', 'fare', 'embarked']: Accuracy = 0.73611
```

Decision tree classification

```
features = ['longitude','latitude']
split = 2
predict = 'category'

# random forest
for x in range(1, 10):
  dt = DecisionTreeClassifier(random_state=0, min_samples_split=split) # split parameter is optional
  dt.fit(citiesTrain[features], citiesTrain[predict])

  predictions = dt.predict(citiesTest[features])
  # print(x ....)

# aggregated predicted output



# Calculate accuracy
actuals = list(citiesTest[predict])
correct = 0
for i in range(len(actuals)):
# print('Predicted:', predictions[i], ' Actual:', actuals[i])
  if predictions[i] == actuals[i]: correct +=1
print('Accuracy:', round(correct/len(actuals),5))
# Try different values for split, different features
```

```
Accuracy: 0.65625
```

"Forest" of decision trees

```
features = ['longitude', 'latitude']
split = 10
trees = 10
predict = 'category'

rf = RandomForestClassifier(random_state=0, min_samples_split=split, n_estimators=trees)
rf.fit(citiesTrain[features], citiesTrain[predict])


predictions = rf.predict(citiesTest[features])
# Calculate accuracy
actuals = list(citiesTest[predict])
correct = 0
for i in range(len(actuals)):
# print('Predicted:', predictions[i], ' Actual:', actuals[i])
  if predictions[i] == actuals[i]: correct +=1
print('Accuracy:', round(correct/len(actuals),5))
# Try different values for split and trees, different features
```

```
Accuracy: 0.78125
```

Your Turn: Decision tree and forest of trees on World Cup data - Graded

```
# SINGLE TREE
# Try different features and different values for split.
# What's the highest accuracy you can get?
# Test different split values

features = ['minutes', 'shots', 'passes', 'tackles', 'saves']
split = 10
predict = 'position'

dt = DecisionTreeClassifier(random_state=0, min_samples_split=split)
dt.fit(playersTrain[features], playersTrain[predict])
predictions = dt.predict(playersTest[features])

actuals = list(playersTest[predict])
correct = 0
for i in range(len(actuals)):
    #print('Predicted:', predictions[i], ' Actual:', actuals[i])
    if predictions[i] == actuals[i]:
        correct += 1

print(f'Min samples split: {split}')
print(f'Accuracy: {round(correct/len(actuals), 5)}')


print("\nTesting different split values")
for split in [2, 5, 10, 15, 20, 25, 30]:
    dt = DecisionTreeClassifier(random_state=0, min_samples_split=split)
    dt.fit(playersTrain[features], playersTrain[predict])
    predictions = dt.predict(playersTest[features])
    actuals = list(playersTest[predict])
    correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
    print(f'Split={split:2d}: Accuracy = {round(correct/len(actuals), 5)}')
```

```
Min samples split: 10
Accuracy: 0.64583

Testing different split values
Split= 2: Accuracy = 0.625
Split= 5: Accuracy = 0.58333
Split=10: Accuracy = 0.64583
```

```
Split=15: Accuracy = 0.6875
Split=20: Accuracy = 0.625
Split=25: Accuracy = 0.625
Split=30: Accuracy = 0.64583
```

```
# FOREST OF TREES
# Try different features and different values for split and trees.
# What's the highest accuracy you can get?

features = ['minutes', 'shots', 'passes', 'tackles', 'saves']
split = 10
trees = 10
predict = 'position'

rf = RandomForestClassifier(random_state=0, min_samples_split=split, n_estimators=trees)
rf.fit(playersTrain[features], playersTrain[predict])
predictions = rf.predict(playersTest[features])

actuals = list(playersTest[predict])
correct = 0
for i in range(len(actuals)):
    #print('Predicted:', predictions[i], ' Actual:', actuals[i])
    if predictions[i] == actuals[i]:
        correct += 1

print(f'Trees: {trees}, Split: {split}')
print(f'Accuracy: {round(correct/len(actuals), 5)}')

# est different configurations
print("\nTesting different configurations")
for trees in [10, 20, 50, 100]:
    for split in [2, 5, 10, 15]:
        rf = RandomForestClassifier(random_state=0, min_samples_split=split, n_estimators=trees)
        rf.fit(playersTrain[features], playersTrain[predict])
        predictions = rf.predict(playersTest[features])
        actuals = list(playersTest[predict])
        correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
        print(f'Trees={trees:3d}, Split={split:2d}: Accuracy = {round(correct/len(actuals), 5)}')
```

```
Trees: 10, Split: 10
Accuracy: 0.6875

Testing different configurations
Trees= 10, Split= 2: Accuracy = 0.75
Trees= 10, Split= 5: Accuracy = 0.625
Trees= 10, Split=10: Accuracy = 0.6875
Trees= 10, Split=15: Accuracy = 0.6875
Trees= 20, Split= 2: Accuracy = 0.72917
Trees= 20, Split= 5: Accuracy = 0.66667
Trees= 20, Split=10: Accuracy = 0.70833
Trees= 20, Split=15: Accuracy = 0.6875
Trees= 50, Split= 2: Accuracy = 0.6875
Trees= 50, Split= 5: Accuracy = 0.6875
Trees= 50, Split=10: Accuracy = 0.70833
Trees= 50, Split=15: Accuracy = 0.70833
Trees=100, Split= 2: Accuracy = 0.70833
Trees=100, Split= 5: Accuracy = 0.6875
Trees=100, Split=10: Accuracy = 0.70833
Trees=100, Split=15: Accuracy = 0.75
```

Your Turn Extra: Decision tree and forest of trees on Titanic data - Graded

```
# SINGLE TREE
# Try different features and different values for split.
# What's the highest accuracy you can get?

features = ['gender', 'age', 'class', 'fare', 'embarked']
split = 10
predict = 'survived'

dt = DecisionTreeClassifier(random_state=0, min_samples_split=split)
dt.fit(titanicTrain[features], titanicTrain[predict])
predictions = dt.predict(titanicTest[features])

actuals = list(titanicTest[predict])
correct = 0
for i in range(len(actuals)):
    # print('Predicted:', predictions[i], ' Actual:', actuals[i])
    if predictions[i] == actuals[i]:
        correct += 1

print(f'Split: {split}')
print(f'Accuracy: {round(correct/len(actuals), 5)}')


print("\nTesting different split values")
for split in [2, 5, 10, 15, 20, 25]:
    dt = DecisionTreeClassifier(random_state=0, min_samples_split=split)
    dt.fit(titanicTrain[features], titanicTrain[predict])
    predictions = dt.predict(titanicTest[features])
    actuals = list(titanicTest[predict])
    correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
    print(f'Split={split:2d}: Accuracy = {round(correct/len(actuals), 5)}')
```

```
Split: 10
Accuracy: 0.80556

Testing different split values
Split= 2: Accuracy = 0.76389
Split= 5: Accuracy = 0.79167
```

```
Split=10: Accuracy = 0.80556
Split=15: Accuracy = 0.76389
Split=20: Accuracy = 0.76389
Split=25: Accuracy = 0.75
```

```
# FOREST OF TREES
# Try different features and different values for split and trees.
# What's the highest accuracy you can get?

features = ['gender', 'age', 'class', 'fare', 'embarked']
split = 10
trees = 10
predict = 'survived'

rf = RandomForestClassifier(random_state=0, min_samples_split=split, n_estimators=trees)
rf.fit(titanicTrain[features], titanicTrain[predict])
predictions = rf.predict(titanicTest[features])

actuals = list(titanicTest[predict])
correct = 0
for i in range(len(actuals)):
    #print('Predicted:', predictions[i], ' Actual:', actuals[i])
    if predictions[i] == actuals[i]:
        correct += 1

print(f'Trees: {trees}, Split: {split}')
print(f'Accuracy: {round(correct/len(actuals), 5)}')


print("\nTesting different configurations")
for trees in [10, 50, 100, 200]:
    for split in [2, 5, 10, 15]:
        rf = RandomForestClassifier(random_state=0, min_samples_split=split, n_estimators=trees)
        rf.fit(titanicTrain[features], titanicTrain[predict])
        predictions = rf.predict(titanicTest[features])
        actuals = list(titanicTest[predict])
        correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
        print(f'Trees={trees:3d}, Split={split:2d}: Accuracy = {round(correct/len(actuals), 5)}')
```

```
Trees: 10, Split: 10
Accuracy: 0.79167

Testing different configurations
Trees= 10, Split= 2: Accuracy = 0.77778
Trees= 10, Split= 5: Accuracy = 0.80556
Trees= 10, Split=10: Accuracy = 0.79167
Trees= 10, Split=15: Accuracy = 0.80556
Trees= 50, Split= 2: Accuracy = 0.76389
Trees= 50, Split= 5: Accuracy = 0.81944
Trees= 50, Split=10: Accuracy = 0.81944
Trees= 50, Split=15: Accuracy = 0.83333
Trees=100, Split= 2: Accuracy = 0.76389
Trees=100, Split= 5: Accuracy = 0.77778
Trees=100, Split=10: Accuracy = 0.81944
Trees=100, Split=15: Accuracy = 0.83333
Trees=200, Split= 2: Accuracy = 0.76389
Trees=200, Split= 5: Accuracy = 0.81944
Trees=200, Split=10: Accuracy = 0.81944
Trees=200, Split=15: Accuracy = 0.83333
```

Naive Bayes classification

```
features = ['longitude', 'latitude']
predict = 'category'

nb = GaussianNB()
nb.fit(citiesTrain[features], citiesTrain[predict])

predictions = nb.predict(citiesTest[features])

# Calculate accuracy
actuals = list(citiesTest[predict])
correct = 0
for i in range(len(actuals)):
# print('Predicted:', predictions[i], ' Actual:', actuals[i])
  if predictions[i] == actuals[i]: correct +=1
print('Accuracy:', round(correct/len(actuals),5))
# Try different features
```

```
Accuracy: 0.78125
```

Your Turn: Naive Bayes on World Cup data

```
# Try different features. What's the highest accuracy you can get?
features = ['minutes', 'shots', 'passes', 'tackles', 'saves']
predict = 'position'

nb = GaussianNB()
nb.fit(playersTrain[features], playersTrain[predict])
predictions = nb.predict(playersTest[features])

actuals = list(playersTest[predict])
correct = 0
for i in range(len(actuals)):
    # print('Predicted:', predictions[i], ' Actual:', actuals[i])
    if predictions[i] == actuals[i]:
        correct += 1
```

```
print(f'Features: {features}')
print(f'Accuracy: {round(correct/len(actuals), 5)}')

# Test different feature combinations
print("\n--- Testing different feature combinations ---")
feature_combos = [
    ['minutes', 'passes'],
    ['minutes', 'shots', 'passes'],
    ['minutes', 'shots', 'passes', 'tackles'],
    ['minutes', 'shots', 'passes', 'tackles', 'saves'],
    ['shots', 'passes', 'tackles', 'saves'],
]

for features in feature_combos:
    nb = GaussianNB()
    nb.fit(playersTrain[features], playersTrain[predict])
    predictions = nb.predict(playersTest[features])
    actuals = list(playersTest[predict])
    correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
    feature_str = ', '.join(features)
    print(f'[{feature_str}]: Accuracy = {round(correct/len(actuals), 5)}')
```

```
Features: ['minutes', 'shots', 'passes', 'tackles', 'saves']
Accuracy: 0.6875

--- Testing different feature combinations ---
[minutes, passes]: Accuracy = 0.45833
[minutes, shots, passes]: Accuracy = 0.5
[minutes, shots, passes, tackles]: Accuracy = 0.66667
[minutes, shots, passes, tackles, saves]: Accuracy = 0.6875
[shots, passes, tackles, saves]: Accuracy = 0.75
```

Your Turn Extra: Naive Bayes on Titanic data - Graded

```
# Try different features. What's the highest accuracy you can get?
features = ['gender', 'age', 'class', 'fare', 'embarked']
predict = 'survived'

nb = GaussianNB()
nb.fit(titanicTrain[features], titanicTrain[predict])
predictions = nb.predict(titanicTest[features])

actuals = list(titanicTest[predict])
correct = 0
for i in range(len(actuals)):
    # print('Predicted:', predictions[i], ' Actual:', actuals[i])
    if predictions[i] == actuals[i]:
        correct += 1

print(f'Features: {features}')
print(f'Accuracy: {round(correct/len(actuals), 5)}')

# Test different feature combinations
print("\n--- Testing different feature combinations ---")
feature_sets = [
    ['gender'],
    ['gender', 'class'],
    ['gender', 'age'],
    ['gender', 'age', 'class'],
    ['gender', 'class', 'fare'],
    ['gender', 'age', 'class', 'fare'],
    ['gender', 'age', 'class', 'fare', 'embarked']
]

for features in feature_sets:
    nb = GaussianNB()
    nb.fit(titanicTrain[features], titanicTrain[predict])
    predictions = nb.predict(titanicTest[features])
    actuals = list(titanicTest[predict])
    correct = sum(1 for i in range(len(actuals)) if predictions[i] == actuals[i])
    feature_str = ', '.join(features)
    print(f'[{feature_str}]: Accuracy = {round(correct/len(actuals), 5)}')
```

```
Features: ['gender', 'age', 'class', 'fare', 'embarked']
Accuracy: 0.76389

--- Testing different feature combinations ---
[gender]: Accuracy = 0.81944
[gender, class]: Accuracy = 0.81944
[gender, age]: Accuracy = 0.81944
[gender, age, class]: Accuracy = 0.81944
[gender, class, fare]: Accuracy = 0.77778
[gender, age, class, fare]: Accuracy = 0.77778
[gender, age, class, fare, embarked]: Accuracy = 0.76389
```

Neural network classification

```
features = ['longitude', 'latitude']
num_layers = 5 # including input and output, so must be >= 2
num_epochs = 10 # number of iterations over training data
batchsize = 20 # size of each batch during one iteration
layer_outputs = 32 # dimensionality of output of each layer
epoch_tracing = 'yes'
predict = 'category'
# Normalize feature values
sc = StandardScaler()
```

```
    featurevals_train = sc.fit_transform(citiesTrain[features])
    featurevals_test = sc.fit_transform(citiesTest[features])
    # Encode labels
    encoder = LabelEncoder()
    encoder.fit(cities[predict])
    labels_train = encoder.transform(citiesTrain[predict])
    labels_test = encoder.transform(citiesTest[predict])
    # Set up neural-net classifier
    seed(1) # to eliminate some randomness
    tensorflow.random.set_seed(1) # to eliminate more randomness
    classifier = Sequential()
    # Input layer
    classifier.add(Dense(layer_outputs, activation='relu', input_dim=len(features)))

    # Hidden layers
    for i in range(num_layers-2):
        classifier.add(Dense(layer_outputs, activation='relu',))


    # Output layer - first arg is number of labels, softmax for multi-class classification
    classifier.add(Dense(4, activation='softmax'))


    classifier.compile(optimizer ='adam', loss='sparse_categorical_crossentropy', metrics =['accuracy'])

    # Fit to training data
    if epoch_tracing == 'yes': v = 2
    else: v = 0
    hist = classifier.fit(featurevals_train, labels_train, batch_size=batchsize, epochs=num_epochs, verbose=v)
    print('Number of epochs:', num_epochs)
    print('Final accuracy on training data:', hist.history['accuracy'][-1])
    # Evaluate on test data
    test_acc = classifier.evaluate(featurevals_test, labels_test, verbose=0)[1]
    print('Accuracy on test data:', test_acc)
    # Try different values for num_layers, num_epochs, batch size, layer_outputs, and different features
```

```
Epoch 1/10
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
10/10 - 2s - 175ms/step - accuracy: 0.4696 - loss: 1.3648
Epoch 2/10
10/10 - 0s - 8ms/step - accuracy: 0.6961 - loss: 1.3136
Epoch 3/10
10/10 - 0s - 13ms/step - accuracy: 0.6630 - loss: 1.2591
Epoch 4/10
10/10 - 0s - 7ms/step - accuracy: 0.6519 - loss: 1.1937
Epoch 5/10
10/10 - 0s - 7ms/step - accuracy: 0.6519 - loss: 1.1177
Epoch 6/10
10/10 - 0s - 7ms/step - accuracy: 0.6685 - loss: 1.0343
Epoch 7/10
10/10 - 0s - 14ms/step - accuracy: 0.6685 - loss: 0.9586
Epoch 8/10
10/10 - 0s - 7ms/step - accuracy: 0.6796 - loss: 0.8968
Epoch 9/10
10/10 - 0s - 7ms/step - accuracy: 0.6906 - loss: 0.8439
Epoch 10/10
10/10 - 0s - 7ms/step - accuracy: 0.6906 - loss: 0.7968
Number of epochs: 10
Final accuracy on training data: 0.6906077265739441
Accuracy on test data: 0.625
```

Your Turn: Neural network on World Cup data

```
# Try different features and different values for num_layers, num_epochs,
#  batch size, and layer_outputs.
# What's the highest accuracy you can get?
# Note: Although some randomness is removed by setting seeds in the code,
#  you may still see somewhat different accuracy on different runs;
#  changing the order of the features can also affect accuracy
features = ['minutes', 'shots', 'passes', 'tackles', 'saves']
num_layers = 5
num_epochs = 50
batchsize = 20
layer_outputs = 64
epoch_tracing = 'no'
predict = 'position'

# Normalize feature values
sc = StandardScaler()
featurevals_train = sc.fit_transform(playersTrain[features])
featurevals_test = sc.transform(playersTest[features])

# Encode labels
encoder = LabelEncoder()
encoder.fit(players[predict])
labels_train = encoder.transform(playersTrain[predict])
labels_test = encoder.transform(playersTest[predict])

# Set up neural-net classifier
seed(1)
tensorflow.random.set_seed(1)
classifier = Sequential()

# Input layer
classifier.add(Dense(layer_outputs, activation='relu', input_dim=len(features)))

# Hidden layers
for i in range(num_layers-2):
```

```
        classifier.add(Dense(layer_outputs, activation='relu'))

    # Output layer - 4 positions
    classifier.add(Dense(4, activation='softmax'))

    classifier.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    # Fit to training data
    if epoch_tracing == 'yes':
        v = 2
    else:
        v = 0

    hist = classifier.fit(featurevals_train, labels_train, batch_size=batchsize, epochs=num_epochs, verbose=v)

    print(f'Layers: {num_layers}, Epochs: {num_epochs}, Batch size: {batchsize}, Layer outputs: {layer_outputs}')
    print(f'Final accuracy on training data: {hist.history["accuracy"][-1]:.5f}')

    # Evaluate on test data
    test_acc = classifier.evaluate(featurevals_test, labels_test, verbose=0)[1]
    print(f'Accuracy on test data: {test_acc:.5f}')

    # Test different configurations
    print("\n--- Testing different epoch values (may take a while) ---")
    for epochs in [10, 30, 50, 100]:
        seed(1)
        tensorflow.random.set_seed(1)
        classifier = Sequential()
        classifier.add(Dense(64, activation='relu', input_dim=len(features)))
        for i in range(3):
            classifier.add(Dense(64, activation='relu'))
        classifier.add(Dense(4, activation='softmax'))
        classifier.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

        hist = classifier.fit(featurevals_train, labels_train, batch_size=20, epochs=epochs, verbose=0)
        test_acc = classifier.evaluate(featurevals_test, labels_test, verbose=0)[1]
        print(f'Epochs={epochs:3d}: Test Accuracy = {test_acc:.5f}')
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Layers: 5, Epochs: 50, Batch size: 20, Layer outputs: 64
Final accuracy on training data: 0.70750
Accuracy on test data: 0.68750

--- Testing different epoch values (may take a while) ---
Epochs= 10: Test Accuracy = 0.70833
Epochs= 30: Test Accuracy = 0.70833
Epochs= 50: Test Accuracy = 0.72917
Epochs=100: Test Accuracy = 0.66667
```

Your Turn Extra: Neural network on Titanic data

```
# Try different features and different values for num_layers, num_epochs,
#  batch size, and layer_outputs.
# What's the highest accuracy you can get?
# Note: Although some randomness is removed by setting seeds in the code,
#  you may still see somewhat different accuracy on different runs;
#  changing the order of the features can also affect accuracy

features = ['gender', 'age', 'class', 'fare', 'embarked']
num_layers = 4
num_epochs = 100
batchsize = 32
layer_outputs = 32
epoch_tracing = 'no'
predict = 'survived'

# Normalize feature values
sc = StandardScaler()
featurevals_train = sc.fit_transform(titanicTrain[features])
featurevals_test = sc.transform(titanicTest[features])

# Encode labels
encoder = LabelEncoder()
encoder.fit(titanic[predict])
labels_train = encoder.transform(titanicTrain[predict])
labels_test = encoder.transform(titanicTest[predict])

# Set up neural-net classifier
seed(1)
tensorflow.random.set_seed(1)
classifier = Sequential()

# Input layer
classifier.add(Dense(layer_outputs, activation='relu', input_dim=len(features)))

# Hidden layers
for i in range(num_layers-2):
    classifier.add(Dense(layer_outputs, activation='relu'))

# Output layer
classifier.add(Dense(2, activation='softmax'))

classifier.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Fit to training data
if epoch_tracing == 'yes':
    v = 2
else:
```

```python
else:
    v = 0

hist = classifier.fit(featurevals_train, labels_train, batch_size=batchsize, epochs=num_epochs, verbose=v)

print(f'Layers: {num_layers}, Epochs: {num_epochs}, Batch size: {batchsize}, Layer outputs: {layer_outputs}')
print(f'Final accuracy on training data: {hist.history["accuracy"][-1]:.5f}')

# Evaluate on test data
test_acc = classifier.evaluate(featurevals_test, labels_test, verbose=0)[1]
print(f'Accuracy on test data: {test_acc:.5f}')

# Test different configurations
print("\nTesting different configurations")
for epochs in [50, 100, 150]:
    for layers in [3, 4, 5]:
        seed(1)
        tensorflow.random.set_seed(1)
        classifier = Sequential()
        classifier.add(Dense(32, activation='relu', input_dim=len(features)))
        for i in range(layers-2):
            classifier.add(Dense(32, activation='relu'))
        classifier.add(Dense(2, activation='softmax'))
        classifier.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

        hist = classifier.fit(featurevals_train, labels_train, batch_size=32, epochs=epochs, verbose=0)
        test_acc = classifier.evaluate(featurevals_test, labels_test, verbose=0)[1]
        print(f'Epochs={epochs:3d}, Layers={layers}: Test Accuracy = {test_acc:.5f}')
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Layers: 4, Epochs: 100, Batch size: 32, Layer outputs: 32
Final accuracy on training data: 0.86203
Accuracy on test data: 0.84722

Testing different configurations
Epochs= 50, Layers=3: Test Accuracy = 0.84722
Epochs= 50, Layers=4: Test Accuracy = 0.83333
Epochs= 50, Layers=5: Test Accuracy = 0.83333
Epochs=100, Layers=3: Test Accuracy = 0.86111
Epochs=100, Layers=4: Test Accuracy = 0.83333
Epochs=100, Layers=5: Test Accuracy = 0.81944
Epochs=150, Layers=3: Test Accuracy = 0.84722
Epochs=150, Layers=4: Test Accuracy = 0.83333
Epochs=150, Layers=5: Test Accuracy = 0.80556
```