13-3

```
# Import required dependencies
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Preprocess data:

df = df.drop(columns='whatever column you want to drop')
df.dtypes – to check data types
df.info()
print(df.columns)
df.reset_index(drop=True, inplace=True) # resets index
IFF the target column has an alphanumeric (not a discrete value); see below.

```
# Drop rows with null values
df_clean = df.dropna().copy()

# Convert y to numeric
df_clean['y'] = pd.get_dummies(df_clean['y'], drop_first = True, dtype=int)

# Drop all non-numeric columns
df_clean = df_clean.select_dtypes(include='number')

# Verify changes with the info method
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5897 entries, 0 to 33904
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       5897 non-null   int64
 1   balance   5897 non-null   int64
 2   day       5897 non-null   int64
 3   duration  5897 non-null   int64
 4   campaign  5897 non-null   int64
 5   pdays     5897 non-null   float64
 6   previous  5897 non-null   int64
 7   y         5897 non-null   uint8
dtypes: float64(1), int64(6), uint8(1)
memory usage: 374.3 KB
```

train_test_split(X,y):

X=df.copy()
X=X.drop(columns='target')
y=df['target']

df['target'].value_counts()
X_train, X_test, y_train, y_test = train_test_split(X,y)

IFF the target is not a discrete value, encode:

```
# Since the target column is an object, we need to convert the data to numerical classes
# Encode the y data
# Create an instance of the label encoder
le = LabelEncoder()

# Fit and transform the y training and testing data using the label encoder
y_train_encoded = le.fit_transform(y_train)
y_test_encoded = le.transform(y_test)
y_train_encoded
```

```
array([2, 2, 0, ..., 0, 2, 0])
```

OneHotEncoder for non-oriented data – colors, favorities

```
# Remember that all of the columns in the DataFrame are objects
# Use a OneHotEncoder to convert the training data to numerical values
ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False, dtype='int')
X_train_encoded = pd.DataFrame(data=ohe.fit_transform(X_train), columns=ohe.get_feature_names_out())
X_train_encoded
```

```
# Encode the test data
X_test_encoded = pd.DataFrame(data=ohe.transform(X_test), columns=ohe.get_feature_names_out())
X_test_encoded
```

OrdinalEncoder for things like grades, days of week

```
# Create an encoder for the grade column – there is a numerical equivalent
grade_ord_enc = OrdinalEncoder(categories = [['F', 'D', 'C', 'B', 'A']], encoded_missing_value=-1,
handle_unknown='use_encoded_value', unknown_value=-1)

# Train the encoder
grade_ord_enc.fit(X_train['grade'].values.reshape(-1,1))
```

```
OrdinalEncoder(categories=[['F', 'D', 'C', 'B', 'A']], encoded_missing_value=-1,
               handle_unknown='use_encoded_value', unknown_value=-1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

xpand output; double click to hide output

Scale if necessary – use when values are different by orders of magnitude:

scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

Note about the X_train and y_train variable.

Some of the X_train and y_train variables are X_train_encode and y_train_encode.
I use X_train_scaled and y_train_scaled.
Same for the X_test/y_test; I use X_test_scaled/y_test_scaled.

Model and Fit to a Support Vector Machine

```
# Create the support vector machine classifier model with a 'poly' kernel
svm_model = SVC(kernel='poly')

# Fit the model to the training data
svm_model.fit(X_train_encoded, y_train_encoded)
```

```
SVC(kernel='poly')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Validate the model by checking the model accuracy with model.score
print('Train Accuracy: %.3f' % svm_model.score(X_train_encoded, y_train_encoded))
print('Test Accuracy: %.3f' % svm_model.score(X_test_encoded, y_test_encoded))
```

```
Train Accuracy: 0.998
Test Accuracy: 0.977
```

Model and Fit to a Logistic Regression Classifier

```
# Create the logistic regression classifier model with a random_state of 1
lr_model = LogisticRegression(random_state=1)

# Fit the model to the training data
lr_model.fit(X_train_encoded, y_train_encoded)
```

```
LogisticRegression(random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Validate the model by checking the model accuracy with model.score
print('Train Accuracy: %.3f' % lr_model.score(X_train_encoded, y_train_encoded))
print('Test Accuracy: %.3f' % lr_model.score(X_test_encoded, y_test_encoded))
```

```
Train Accuracy: 0.925
Test Accuracy: 0.917
```

Model and Fit to a Decision Tree Classifier

```
# Create the decision tree classifier model
dt_model = DecisionTreeClassifier()

# Fit the model to the training data
dt_model.fit(X_train_encoded, y_train_encoded)
```

```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Validate the model by checking the model accuracy with model.score
print('Train Accuracy: %.3f' % dt_model.score(X_train_encoded, y_train_encoded))
print('Test Accuracy: %.3f' % dt_model.score(X_test_encoded, y_test_encoded))
```

```
Train Accuracy: 1.000
Test Accuracy: 0.968
```

Model and Fit to a Random Forest Classifier

```
# Create the random forest classifier model
# with n_estimators=128 and random_state=1
rf_model = RandomForestClassifier(n_estimators=128, random_state=1)

# Fit the model to the training data
rf_model.fit(X_train_encoded, y_train_encoded)
```

```
RandomForestClassifier(n_estimators=128, random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Validate the model by checking the model accuracy with model.score
print('Train Accuracy: %.3f' % rf_model.score(X_train_encoded, y_train_encoded))
print('Test Accuracy: %.3f' % rf_model.score(X_test_encoded, y_test_encoded))
```

```
Train Accuracy: 1.000
Test Accuracy: 0.961
```

See next page to optimize the max_depth of the RFC model

Optimize the hyperparameter "max_depth" for the Random Forest Classifier

```
# Try the following values for max_depth

max_depths = range(1, 10)
models = {'train_score': [], 'test_score': [], 'max_depth': []}

# Loop through each value in max_depths
for depth in max_depths:
    clf = RandomForestClassifier(max_depth = depth)
    clf.fit(X_train, y_train)

    train_pred = clf.predict(X_train)
    test_pred = clf.predict(X_test)

    train_score = balanced_accuracy_score(y_train, train_pred)
    test_score = balanced_accuracy_score(y_test, test_pred)

    models['train_score'].append(train_score)
    models['test_score'].append(test_score)
    models['max_depth'].append(depth)

# Create a dataframe from the models dictionary with max_depth as the index
models_df = pd.DataFrame(models).set_index('max_depth')
```

```
# Plot the results
models_df.plot()
```

```
clf = RandomForestClassifier(max_depth=5)
clf.fit(X_train, y_train)

train_pred = clf.predict(X_train)
test_pred = clf.predict(X_test)

print(balanced_accuracy_score(y_train, train_pred))
print(balanced_accuracy_score(y_test, test_pred))
```

```
0.6322281365725427
0.5937545762574925
```

```
# Check the model's balanced accuracy on the test set

y_pred = model.predict(X_test)
print(balanced_accuracy_score(y_test, y_pred))
```

```
0.5790651053107311
```

```
# Check the model's balanced accuracy on the training set

y_train_pred = model.predict(X_train)
print(balanced_accuracy_score(y_train, y_train_pred))
```

```
0.6257478761408065
```

KNN without optimization

```
# Create the KNN model with 9 neighbors
knn_model = KNeighborsClassifier(n_neighbors=9)

# Fit the model to the training data
knn_model.fit(X_train_encoded, y_train_encoded)
```

```
KNeighborsClassifier(n_neighbors=9)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Validate the model by checking the model accuracy with model.score
print('Train Accuracy: %.3f' % knn_model.score(X_train_encoded, y_train_encoded))
print('Test Accuracy: %.3f' % knn_model.score(X_test_encoded, y_test_encoded))
```

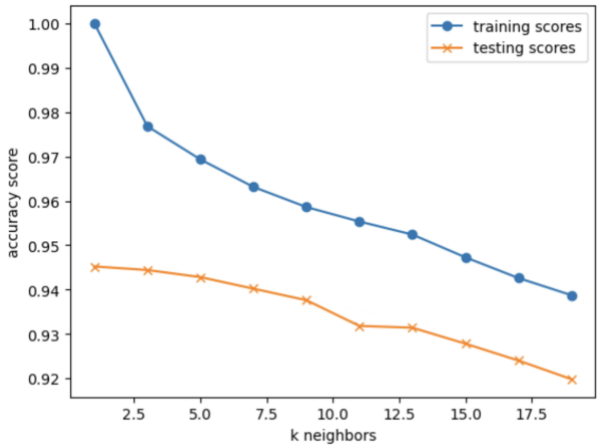
```
Train Accuracy: 0.960
Test Accuracy: 0.919
```

Model and Fit to a KNN Model

```
import matplotlib.pyplot as plt
# Loop through different k values to find which has the highest accuracy.
# Note: We use only odd numbers because we don't want any ties.
train_scores = []
test_scores = []
for k in range(1, 20, 2):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train_encoded)
    train_score = knn.score(X_train_scaled, y_train_encoded)
    test_score = knn.score(X_test_scaled, y_test_encoded)
    train_scores.append(train_score)
    test_scores.append(test_score)
    print(f"k: {k}, Train/Test Score: {train_score:.3f}/{test_score:.3f}")

# Plot the results
plt.plot(range(1, 20, 2), train_scores, marker='o', label="training scores")
plt.plot(range(1, 20, 2), test_scores, marker="x", label="testing scores")
plt.xlabel("k neighbors")
plt.ylabel("accuracy score")
plt.legend()
plt.show()
```

```
k: 1, Train/Test Score: 1.000/0.945
k: 3, Train/Test Score: 0.977/0.944
k: 5, Train/Test Score: 0.969/0.943
k: 7, Train/Test Score: 0.963/0.940
k: 9, Train/Test Score: 0.959/0.938
k: 11, Train/Test Score: 0.955/0.932
k: 13, Train/Test Score: 0.952/0.931
k: 15, Train/Test Score: 0.947/0.928
k: 17, Train/Test Score: 0.943/0.924
k: 19, Train/Test Score: 0.939/0.920
```




```
# Create the KNN model with 3 neighbors
knn_model = KNeighborsClassifier(n_neighbors=3)

# Fit the model to the training data
knn_model.fit(X_train_scaled, y_train_encoded)
```

```
KNeighborsClassifier(n_neighbors=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Validate the model by checking the model accuracy with model.score
print(f"Training Data Score: {knn_model.score(X_train_scaled, y_train_encoded)}")
print(f"Testing Data Score: {knn_model.score(X_test_scaled, y_test_encoded)}")
```

```
Training Data Score: 0.9768
Testing Data Score: 0.9444
```

Model and Fit to a Gradient Boosting Classifier

```
# Train the Gradient Boosting classifier
clf = GradientBoostingClassifier(random_state=1).fit(X_train_scaled, y_train_encoded)

# Evaluate the model
print(f'Training Score: {clf.score(X_train_scaled, y_train_encoded)}')
print(f'Testing Score: {clf.score(X_test_scaled, y_test_encoded)}')
```

```
Training Score: 0.9622666666666667
Testing Score: 0.9152
```

Model and Fit to an Adaptive Boosting Classifier

```
# Train the AdaBoostClassifier
clf = AdaBoostClassifier(random_state=1).fit(X_train_scaled, y_train_encoded)

# Evaluate the model
print(f'Training Score: {clf.score(X_train_scaled, y_train_encoded)}')
print(f'Testing Score: {clf.score(X_test_scaled, y_test_encoded)}')
```

```
Training Score: 0.274
Testing Score: 0.2776
```

13-3-4: Familiar Regressors:

```
%matplotlib inline
from matplotlib import pyplot as plt
from sklearn.datasets import make_regression, make_swiss_roll
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
def test_model(model, data):
    X_train_scaled, X_test_scaled, y_train, y_test = data
    reg = model.fit(X_train_scaled, y_train)
    print(f'Model: {type(reg).__name__}')
    print(f'Train score: {reg.score(X_train_scaled, y_train)}')
    print(f'Test Score: {reg.score(X_test_scaled, y_test)}\n')
    plt.show()
```

```
# Create data
X, y = make_regression(random_state=1)
X = pd.DataFrame(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
data = [X_train_scaled, X_test_scaled, y_train, y_test]
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor, AdaBoostRegressor
from sklearn.svm import SVR
```

```
test_model(LinearRegression(), data)

test_model(KNeighborsRegressor(), data)
test_model(RandomForestRegressor(), data)
test_model(ExtraTreesRegressor(), data)
test_model(AdaBoostRegressor(), data)
test_model(SVR(C=1.0, epsilon=0.2), data)
```

```
Model: LinearRegression
Train score: 1.0
Test Score: 0.8288596458298328

Model: KNeighborsRegressor
Train score: 0.5300655721367826
Test Score: 0.07078594554559592

Model: RandomForestRegressor
Train score: 0.9081964906139426
Test Score: 0.28468630340953627

Model: ExtraTreesRegressor
Train score: 1.0
Test Score: 0.46765205432533674

Model: AdaBoostRegressor
Train score: 0.9321517489091534
Test Score: 0.3563551886522288

Model: SVR
Train score: 0.006215310460808365
Test Score: -0.010118247516417211
```

```python
# Plot the result
ax = plt.figure().add_subplot(projection='3d')
ax.view_init(7, -80)
ax.scatter(X[0], X[1], X[2],
           color=plt.cm.jet(y/y.max()),
           s=20, edgecolor='k')
plt.savefig("swiss_roll.png")
plt.show()
```