Abalone Nearest Neighbor Model

Author: Charlie McCollough

The below code is to import the abalone data set and print out the abalone table to ensure that the columns rows and variables were correct.

In [42]:
```
import matplotlib .pyplot as plt
import numpy as np
import pandas as pd
import mglearn

#import data set
column_names = ["sex", "length", "diameter", "height", "whole weight",
"shucked weight", "viscera weight", "shell weight", "rings"
]
abalone= pd.read_csv("abalone.data", names=column_names )
print("Number of samples: %d" % len(abalone))
abalone.head ()
for label in "MFI":
    abalone[label] = abalone["sex"] == label
del abalone["sex"]
X = abalone.drop("rings", axis=1)
X = X.values
y = abalone["rings"]
y = y.values

abalone
```

```
Number of samples: 4177
```

Out[42]:

	length	diameter	height	whole weight	shucked weight	viscera weight	shell weight	rings	M	F	I
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15	True	False	False
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7	True	False	False
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9	False	True	False
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10	True	False	False
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...
4172	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11	False	True	False
4173	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10	True	False	False
4174	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9	True	False	False
4175	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10	False	True	False
4176	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12	True	False	False

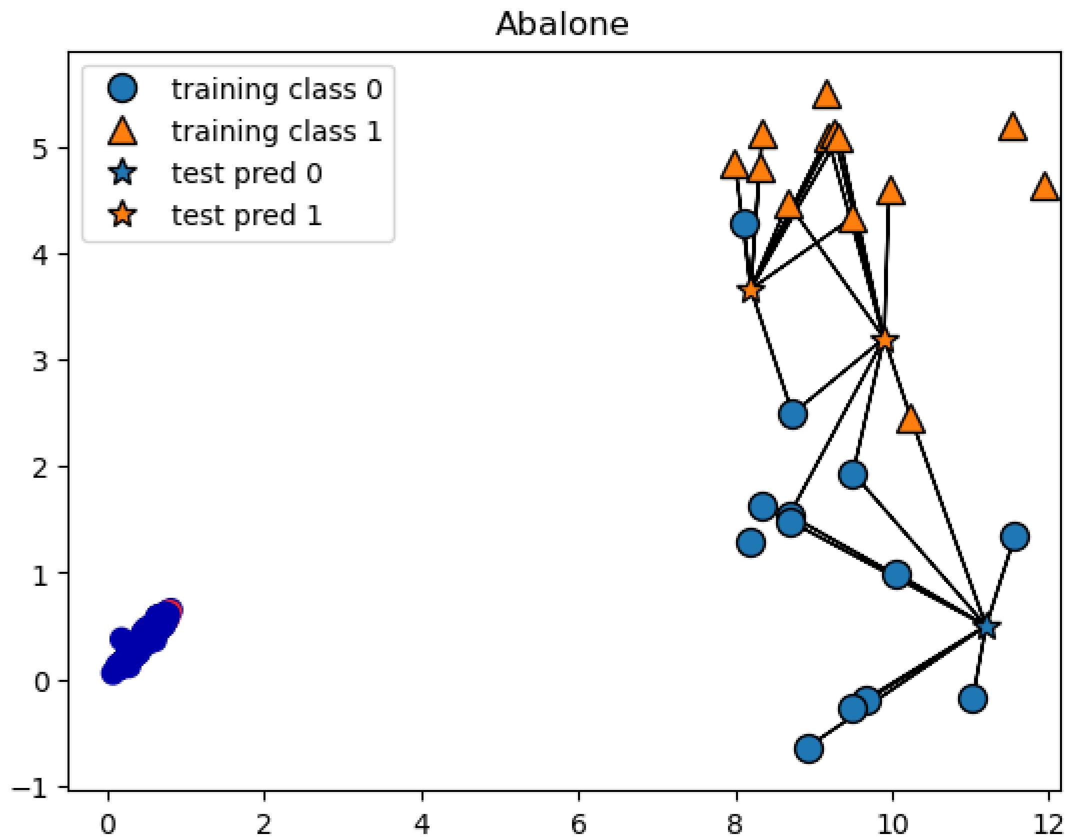
4177 rows × 11 columns

To plot the abalone data set using the neareest neighbor approach we set the neighbor count to 10 and plotted using mglearn.

In [43]:

```
#plot for abalone
plt.scatter(X[:, 0], X[:, 1], c=y, s=60, cmap=mglearn.cm2)
print("X.shape: %s" % (X.shape,))

#make sure this is right******************************
mglearn.plots.plot_knn_classification(n_neighbors=10)
plt.title("Abalone");
```

```
X.shape: (4177, 10)
```



In this the training test split for the abalone dataset. We set it into X train, Y train, X test, and Y test. Using these splits we can use the KNeighbors regressor to predict the model and allow the test data to test the model and find the R squared value and t test statistic. I also plotted the test and training accuracy for the model in the plot below to show the relationship between the train and test data sets.

In [48]:

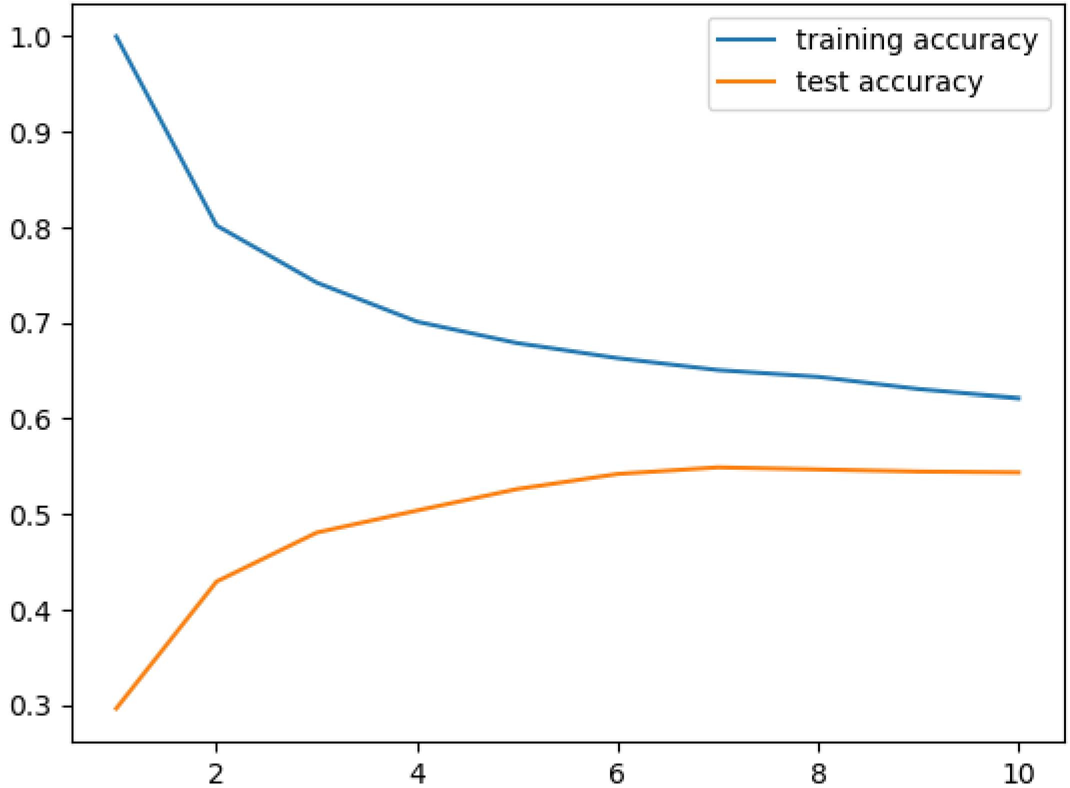
```
import matplotlib .pyplot as plt
import numpy as np
import pandas as pd
import mglearn
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

Xtr, Xtest, ytr, ytest = train_test_split(X, y, test_size=0.2, random_state=42)
#change neighbor count to get highest R^2 value 108 gives .543)
k_neighbors = 10
```

```python
knn_model = KNeighborsRegressor(n_neighbors=k_neighbors)
knn_model.fit(Xtr, ytr)
yp = knn_model.predict(Xtest)

meansquarederror = mean_squared_error(ytest, yp)
rsquared = r2_score(ytest, yp)
print(f"Mean Squared Error (MSE): {meansquarederror}")
print(f"R-squared (R2) Score: {rsquared}")

training_accuracy = []
test_accuracy = []
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
    knn_model = KNeighborsRegressor(n_neighbors=n_neighbors)
    knn_model.fit(Xtr, ytr)
    training_accuracy.append(knn_model.score(Xtr, ytr))
    test_accuracy.append(knn_model.score(Xtest, ytest))
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.legend()
```

```
Mean Squared Error (MSE): 4.942416267942584
R-squared (R2) Score: 0.5434346079990524
```

Out[48]:

```
<matplotlib.legend.Legend at 0x1f55ef099f0>
```



My reasoning for using 10 neighbors is that it gives the biggest R^2 value for the set. The R value being the highest means that the goodness of fit is the highest with 10 as the neighbor count.

These are the linear regression intercept and coefficients for each of the variables in the abalone set.

In [27]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression().fit(Xtr, ytr)
print("lr.coef_: %s" % lr.coef_)
print("lr.intercept_: %s" % lr.intercept_)
```

```
lr.coef_: [ -0.20155385  11.12339118  10.44532535   8.93217555 -20.25654479
  -9.5589163    8.79237823   0.3085089    0.20523277  -0.51374167]
lr.intercept_: 3.5382374338769518
```

These are the training and test scores for the model, using both ridge and regular regressions.

In [52]:

```
from sklearn.linear_model import RidgeCV
ridge_cv = RidgeCV(alphas=[0.01, 0.1, 1.0, 10.0], cv=5)
ridge_cv.fit(Xtr, ytr)
best_alpha = ridge_cv.alpha_
ridge_model = Ridge(alpha=best_alpha).fit(Xtr, ytr)
test_score = ridge_model.score(Xtest, ytest)
print(f"Test R-squared: {test_score}")
```

```
Test R-squared: 0.5449349701624066
```

In [28]:

```
print("training set score: %f" % lr.score(Xtr, ytr))
print("test set score: %f" % lr.score(Xtest, ytest))
```

```
training set score: 0.534824
test set score: 0.548163
```

In [ ]: