schulzd_Lab3

September 28, 2020

1 Lab 3 - KNN

David Schulz

1.1 Reflection Questions

1.1.1 Problem 1

1. Estimate the run time complexity in big-o notation of training a KNN model. Justify your answer.
   - "Training" is just storing the points and their labels in the class, so it's just O(1).
2. Estimate the run time complexity in big-o notation of predicting the output value for a query point using a training set of N points, with p features, and k neighbors with a KNN model. Justify your answer.
   - Computing distances is O(Np), sorting the list is O(k log N), getting the known labels is O(k), and aggregation is O(k), so O(Np + k log N + k) = O(Np).
3. What do you think the potential downsides of the k nearest neighbors algorithm are? Why do you think it might not be used as widely as other methods?
   - The entire training set must be stored, query run time is slower than most other models, it's very sensitive to feature scaling, and there's no feature weighting. Most other methods are much faster and allow for more advanced techniques like feature weighting.

1.1.2 Problem 2

1. For each of the three data sets, do you think a linear decision boundary could be used to accurately classify the data points?
   - A linear decision boundary could have been used for the moon and rocky ridge data sets because the classes seem to be split fairly linear, with the exception of some bulges in the moon plot. A linear decision boundary should not be used for the circles data set because the classes are split with one in a circle and the other surrounding it.
2. What do we mean by a "non-linear" decision boundary? Give an example of a non-linear function that could be used as a decision boundary for one of the data sets.
   - It could be any equation that accurately separates the two classes. For example, an equation that creates a circle, such as $x^{2+y}2=1$, could be used to split the classes for the circles dataset.

3. What are the advantages of non-linear decision boundaries over linear decision boundaries? What are the disadvantages?
    - The non-linear decision boundary can fit the data better, allowing more accurate classification. It can, however, also lead to overfitting, where the decision boundary begins to just memorize the training points.

1.1.3 Problem 3

1. What value of k gave the highest accuracy?
    - It appears that k values around 75 gave the best accuracy.
2. For large values of k, what happened to the accuracy? Why do you think this is?
    - The accuracy dropped to 0.5 because at that point it was just checking all of the reference points and the chance of classifying correctly was 50/50.
3. Let's say that we ask you to use the following experimental setup to find a value of k that maximizes the accuracy of the model's predictions: split the data set into training and testing sets, train a model for each value of k using the training set, predictions the classes of the testing points, and evaluate the accuracy of the predictions. Could this approach give you misleading results? If so, why?
    - Not necessarily because the data being used to test wasn't used to train. It may still give an accuracy that's worse than what it would be if cross-fold validation was used.
4. It is considered a "best practice" to use cross-fold validation when optimizing parameters. Why do you think that is?
    - It allows you to consider multiple different possible combinations of your given dataset so that no reference points are left out and some of the uncertainty can be diminished.

1.2 1. Implement the kNN Algorithm

[1]:
```
!python test_knn.py
```

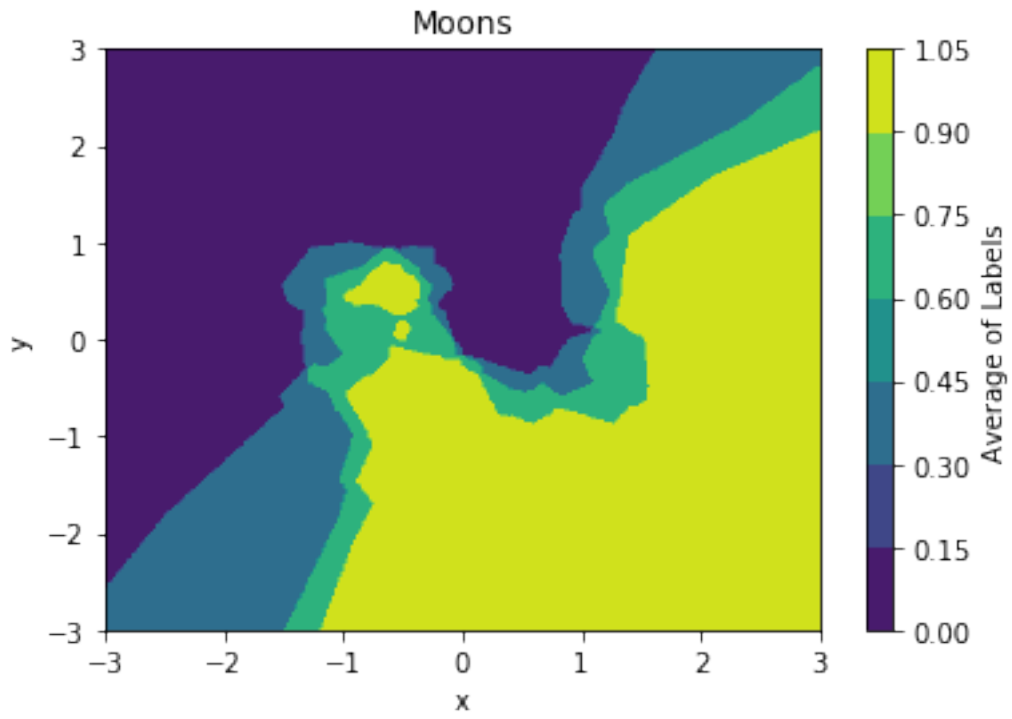
```
…
----------------------------------------------------------------------
Ran 3 tests in 0.032s

OK
```

1.3 2. Explore Decision Boundaries for 3 Data Sets

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from knn import KNN

moons = pd.read_csv('moons.csv')
```

```
y = moons['label'].to_numpy()
X = moons.drop('label', axis=1).to_numpy()

train_X, test_X, train_y, test_y = train_test_split(X, y, stratify=y)

scaler = StandardScaler()
train_X = scaler.fit_transform(train_X)
test_X = scaler.transform(test_X)

knn = KNN(3, "average")
knn.fit(train_X, train_y)

xs = np.linspace(-3, 3, num=300)
points = np.meshgrid(xs, xs)
xs = points[0].reshape((-1, 1))
ys = points[1].reshape((-1, 1))
test_X = np.hstack((xs, ys))
pred_y = knn.predict(test_X)

plt.title('Moons')
plt.xlabel('x')
plt.ylabel('y')
cs = plt.contourf(points[0], points[1], pred_y.reshape((len(points[1]),␣
 ↪len(points[0]))))
cbar = plt.colorbar(cs)
cbar.ax.set_ylabel('Average of Labels')
```

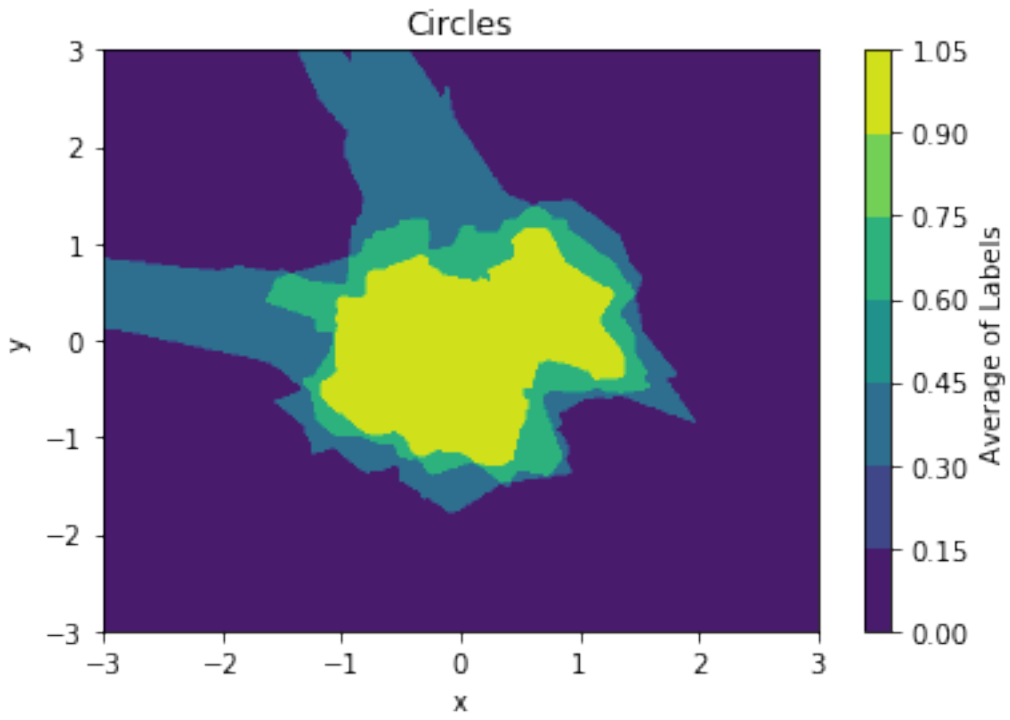
[2]: Text(0, 0.5, 'Average of Labels')



[3]:
```
circles = pd.read_csv('circles.csv')

y = circles['label'].to_numpy()
X = circles.drop('label', axis=1).to_numpy()

train_X, test_X, train_y, test_y = train_test_split(X, y, stratify=y)

scaler = StandardScaler()
train_X = scaler.fit_transform(train_X)
test_X = scaler.transform(test_X)

knn = KNN(3, "average")
knn.fit(train_X, train_y)

xs = np.linspace(-3, 3, num=300)
points = np.meshgrid(xs, xs)
xs = points[0].reshape((-1, 1))
ys = points[1].reshape((-1, 1))
test_X = np.hstack((xs, ys))
pred_y = knn.predict(test_X)

plt.title('Circles')
plt.xlabel('x')
```

```
plt.ylabel('y')
cs = plt.contourf(points[0], points[1], pred_y.reshape((len(points[1]),␣
 ↪len(points[0]))))
cbar = plt.colorbar(cs)
cbar.ax.set_ylabel('Average of Labels')
```

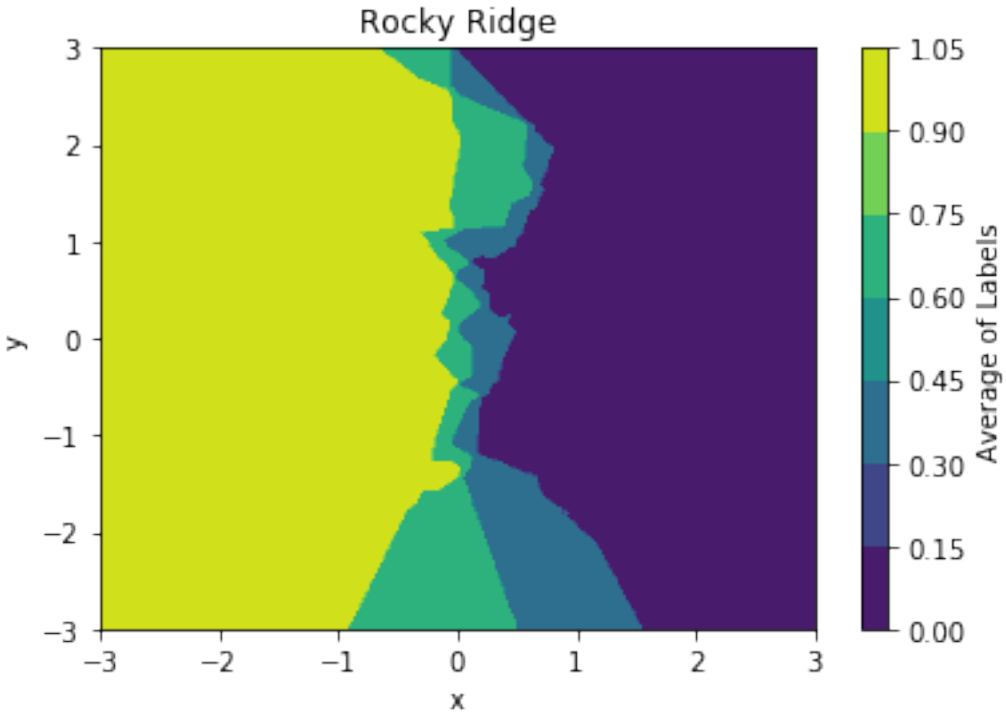
[3]: Text(0, 0.5, 'Average of Labels')



[4]:
```
rocky = pd.read_csv('rocky_ridge.csv')

y = rocky['label'].to_numpy()
X = rocky.drop('label', axis=1).to_numpy()

train_X, test_X, train_y, test_y = train_test_split(X, y, stratify=y)

scaler = StandardScaler()
train_X = scaler.fit_transform(train_X)
test_X = scaler.transform(test_X)

knn = KNN(3, "average")
knn.fit(train_X, train_y)

xs = np.linspace(-3, 3, num=300)
```

```
points = np.meshgrid(xs, xs)
xs = points[0].reshape((-1, 1))
ys = points[1].reshape((-1, 1))
test_X = np.hstack((xs, ys))
pred_y = knn.predict(test_X)

plt.title('Rocky Ridge')
plt.xlabel('x')
plt.ylabel('y')
cs = plt.contourf(points[0], points[1], pred_y.reshape((len(points[1]),␣
↪len(points[0]))))
cbar = plt.colorbar(cs)
cbar.ax.set_ylabel('Average of Labels')
```

[4]: Text(0, 0.5, 'Average of Labels')

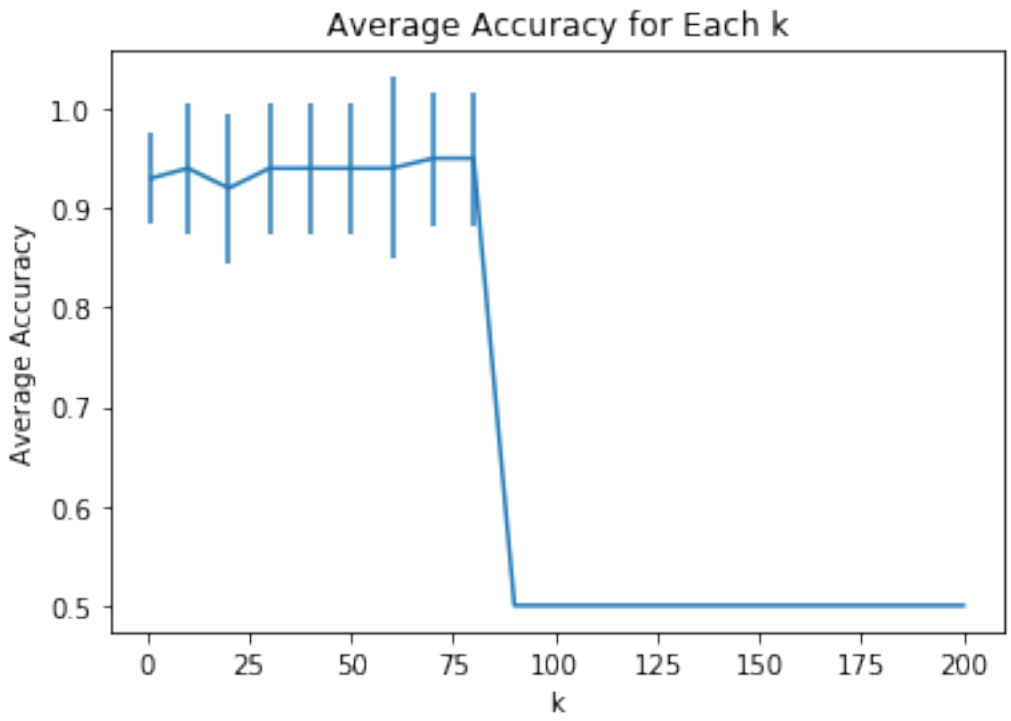


1.4 3. Choosing an Optimal Value for k

[8]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score

sweep = pd.read_csv('sweep.csv')
```

```
y = rocky['label'].to_numpy()
X = rocky.drop('label', axis=1).to_numpy()

avgs = []
stds = []

for k in range(0, 201):
    if k % 10 != 0:
        continue
    if k == 0:
        k = 1

    knn = KNN(k, "mode")
    accuracies = np.ndarray((10))

    skf = StratifiedKFold(n_splits=10, shuffle=True)
    i = 0
    for train_index, test_index in skf.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        knn.fit(X_train, y_train)
        pred_y = knn.predict(X_test)
        accuracies[i] = accuracy_score(y_test, pred_y)
        i += 1

    avgs.append([k, np.mean(accuracies)])
    stds.append([k, np.std(accuracies)])

avgs = np.array(avgs)
stds = np.array(stds)

plt.title('Average Accuracy for Each k')
plt.xlabel('k')
plt.ylabel('Average Accuracy')
plt.errorbar(avgs[:, 0], avgs[:, 1], yerr=stds[:, 1])
```

[8]: <ErrorbarContainer object of 3 artists>


Average Accuracy for Each k
Average Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
25
50
75
100
125
150
175
200
k