14. K-Nearest Neighbors (KNN)

Bruce E. Shapiro

Getting Started in Machine Learning

Last revised: March 20, 2019

KNN-Theory

- Bayes' Rule $P(c_k|\mathbf{x}) = \dfrac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})}$

KNN-Theory

- Bayes' Rule $P(c_k|\mathbf{x}) = \dfrac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})}$
- Prior Probability: $p(c_k) = \dfrac{N_k}{N}$
  - N_k, N=class k, total in data set

KNN-Theory

- Bayes' Rule $P(c_k|\mathbf{x}) = \dfrac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})}$
- Prior Probability: $p(c_k) = \dfrac{N_k}{N}$
  - N_k, N=class k, total in data set
- Conditional Probabilty: $p(\mathbf{x}|c_k) = \dfrac{n_k/N_k}{V} = \dfrac{n_k}{N_k V}$
  - n_k=class k in ball of volume V centered at $\mathbf{x}$

KNN-Theory

- Bayes' Rule $P(c_k|\mathbf{x}) = \dfrac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})}$
- Prior Probability: $p(c_k) = \dfrac{N_k}{N}$
  - N_k, N=class k, total in data set
- Conditional Probabilty: $p(\mathbf{x}|c_k) = \dfrac{n_k/N_k}{V} = \dfrac{n_k}{N_k V}$
  - n_k=class k in ball of volume V centered at $\mathbf{x}$
- Unconditional Probability: $p(\mathbf{x}) = \dfrac{K/N}{V} = \dfrac{K}{NV}$

KNN Algorithm

- Bayes' Rule gives:

$$p(c_k|\mathbf{x}) = \frac{p(\mathbf{x}|c_k)p(c_k)}{p(\mathbf{x})} = \frac{\frac{n_k}{N_k V}\frac{N_k}{N}}{\frac{K}{NV}} = \frac{n_k}{K}$$

- Algorithm becomes:

$$y_p = \arg\max_{c_k} \; P(c_k|\mathbf{x}) = \arg\max_{c_k} \; n_k$$

Find the classes that maximize the probability, i.e., a point $\mathbf{x}$ is assigned to the class that has the most points in it within a unit ball of size V. The ball is defined by the number of near neighbors.

KNN in Python (Example)

Create a 3-class file (extract 4,6,8 cylinder data):

```
cardata=np.array(data[["cyl","mpg","accel"]])
cars=np.array([line for line in cardata
   if line[0] in [4,6,8]])

X=cars[:,1:] # columns 1 and 2, mpg and accel

cylinders=np.unique(cars[:,0])
Y=np.array([float(cylinders.tolist().index(j))
  for j in cars[:,0]]) # 0, 1, 2 for 4, 6, 8 cylinders
```

KNN in Python (Example)

Create a 3-class file (extract 4,6,8 cylinder data):

```
cardata=np.array(data[["cyl","mpg","accel"]])
cars=np.array([line for line in cardata
   if line[0] in [4,6,8]])

X=cars[:,1:] # columns 1 and 2, mpg and accel

cylinders=np.unique(cars[:,0])
Y=np.array([float(cylinders.tolist().index(j))
  for j in cars[:,0]]) # 0, 1, 2 for 4, 6, 8 cylinders
```

Imports:

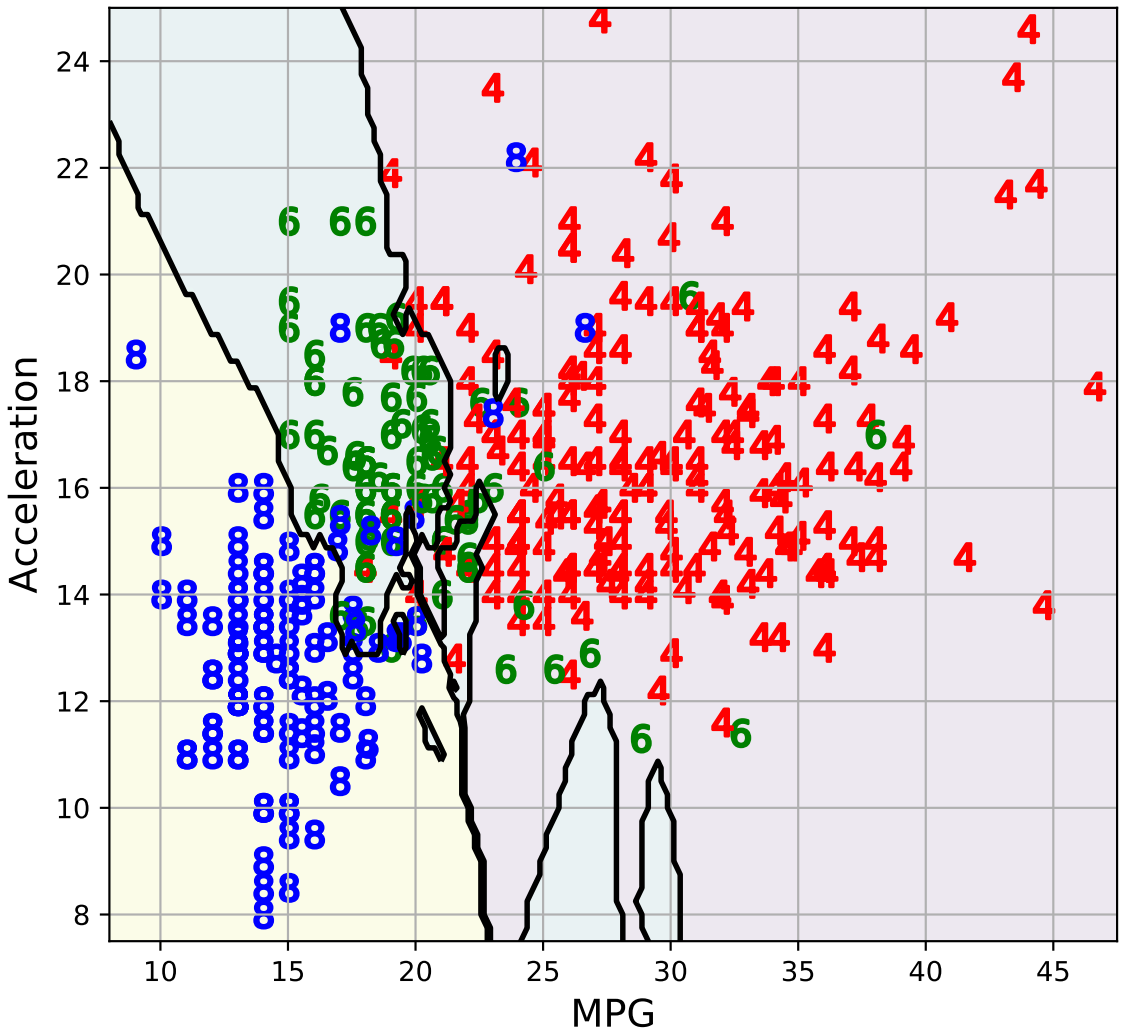
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
```

Results of 100 Runs with 4 near neighbors

```
nruns = 100
errs = []
for j in range(nruns):
    KNN=KNeighborsClassifier(n_neighbors=4)
    XTRAIN, XTEST, YTRAIN, YTEST=train_test_split(X,Y)
    KNN.fit(XTRAIN, YTRAIN)
    YP=KNN.predict(XTEST)
    errs.append(1-accuracy_score(YTEST,YP))
print("%d Splits: Mean Error=%7.6f +/- %7.6f (95%%)"\
      %(nruns, np.mean(errs),1.96*np.std(errs)))
print("Final confusion_matrix:")
print(confusion_matrix(YTEST,YP))
```

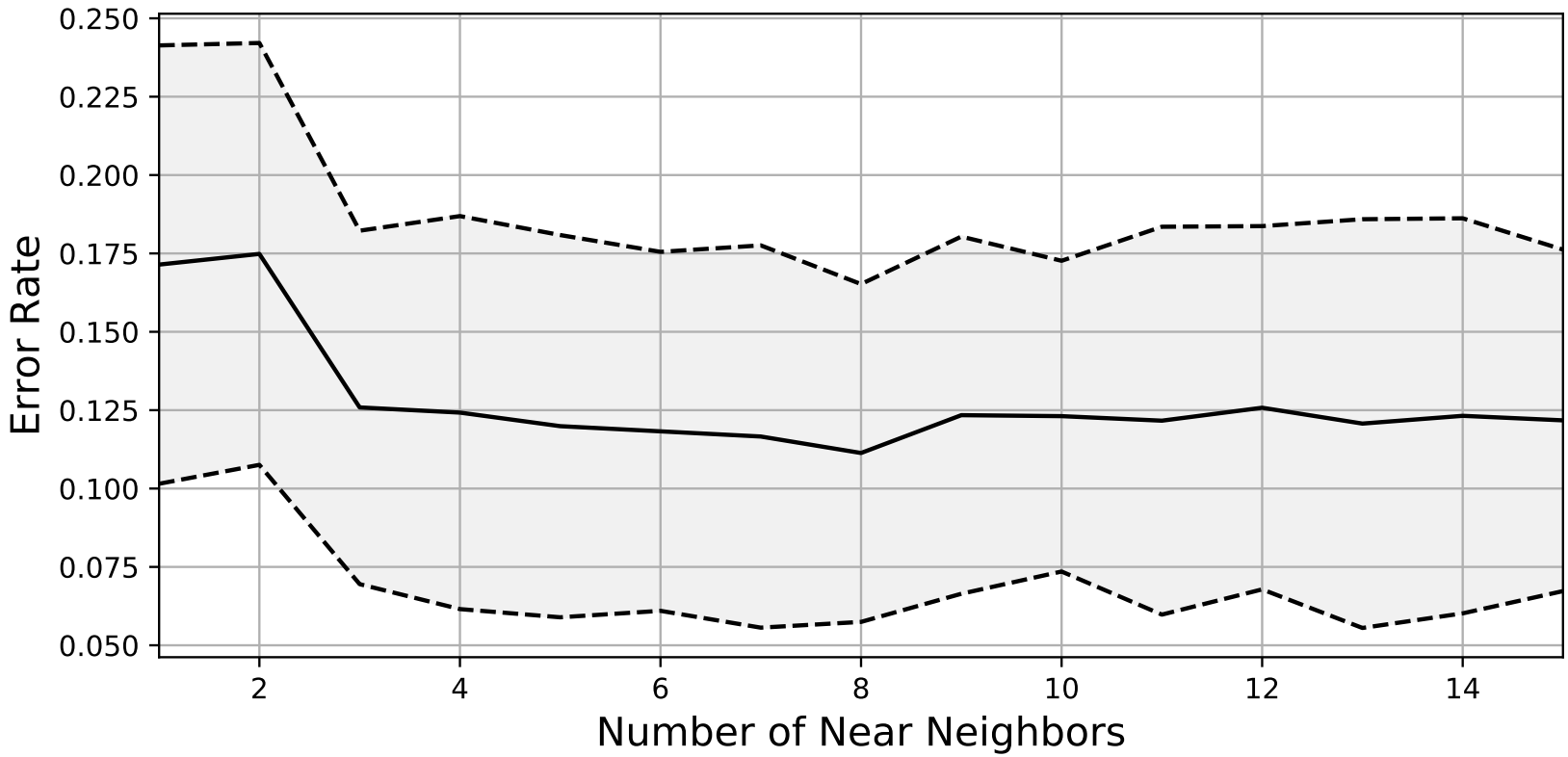
```
100 Splits: Mean Error=0.120103 +/- 0.060171 (95%)
Final confusion_matrix:
[[44  3  0]
 [ 7 17  0]
 [ 1  4 21]]
```

4-Near Neighbor Decision Boundary

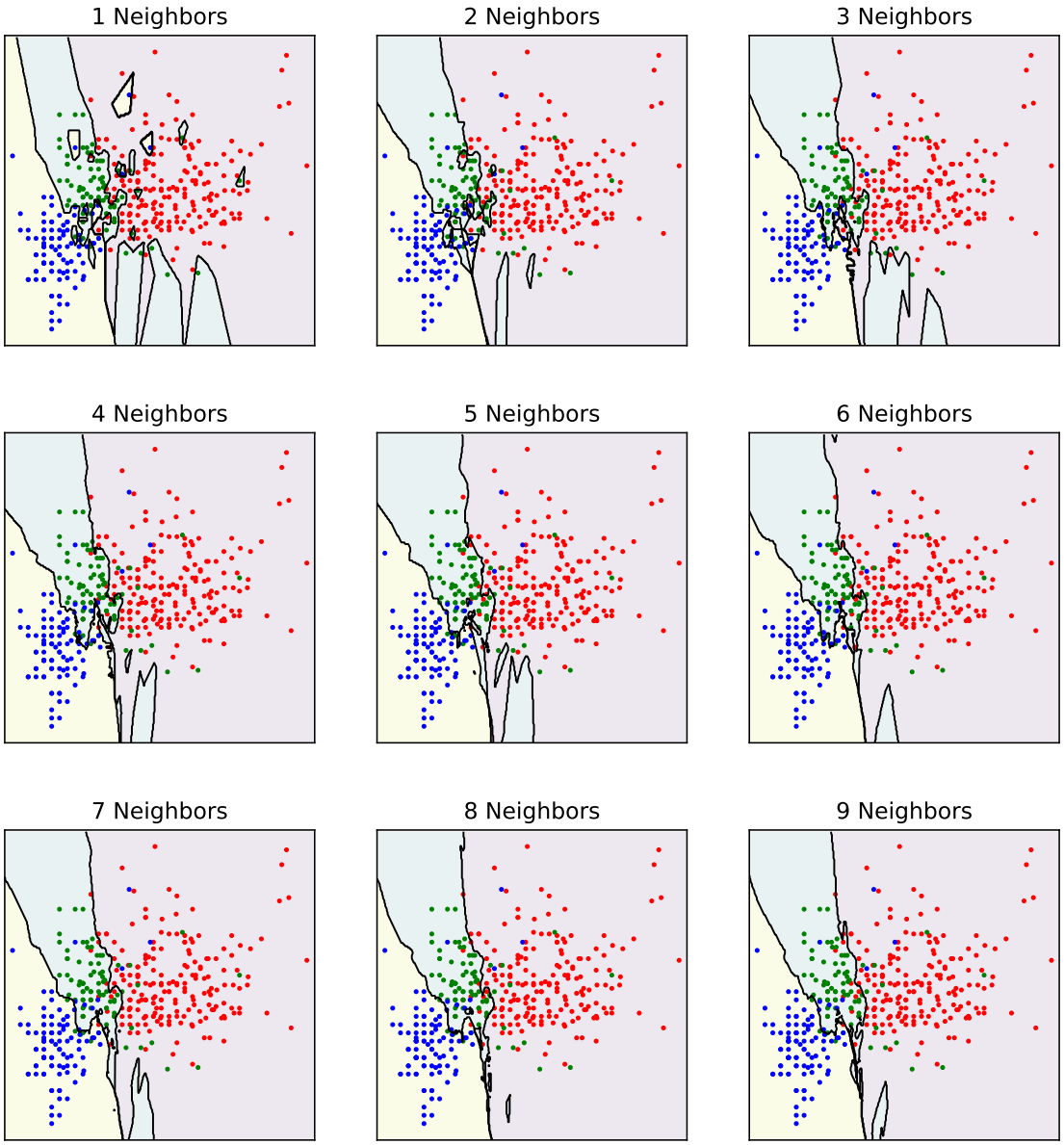


How Many Neighbors?

100 train/test splits on predicting cylinders, with 2 features



How Many Neighbors?



Increasing Number of Features

4 Features, 5 Classes (3, 4, 5, 6, or 8 Cylinders)

```
X=np.array(data[["mpg","displ","hp","accel"]])
Y=np.array(data["cyl"])
unis=np.unique(Y).tolist()
...
# code omitted for produce 100 train/test splits
```

```
100 Splits: Mean Error=0.036531 +/- 0.032641 (95%)
```

The error decreases from 12% to 3.7%

Add origin as 1-Hot Encoding

7 Features, 5 classes

```
origin=np.array(data["origin"])
amer=np.array([1 if j==1 else 0 for j in origin]).reshape(-1,1
euro=np.array([1 if j==2 else 0 for j in origin]).reshape(-1,1
asia=np.array([1 if j==3 else 0 for j in origin]).reshape(-1,1
X=np.array(data[["mpg","displ","hp","accel"]])
X=np.hstack((X,amer,euro,asia))
...
# code omitted for produce 100 train/test splits
```

```
100 Splits: Mean Error=0.039796 +/- 0.034814 (95%)
```

It doesn't help, but at least it doesn't hurt!

References

1. MPG data from: Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository `http://archive.ics.uci.edu/ml`. Irvine, CA: University of California, School of Information and Computer Science.