ISTA 421/521 – Homework 5

Due: Wednesday, December 5, 5pm

20 pts total for Undergrads, 24 pts total for Grads

Ken Youens-Clark

Graduate

N.B.: I worked with Kai Blumberg, Matt Miller, and Enrico Schiassi.

1. [5 points] Adapted from **Exercise 5.3** of FCMA p.202:

   Compute the maximum likelihood estimates of $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ for class c of a Bayesian classifier with Gaussian class-conditionals and a set of N_c objects belonging to class c, $\mathbf{x}_1$, ..., $\mathbf{x}_{N_c}$.

   Solution.

   Likelihood

   $$L = p(\mathbf{X}^c|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

   is product of (Gaussian) probabilities

   $$= \prod_n^N \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

   $$= \prod_n^N \frac{1}{(2\pi)^{N/2}|\boldsymbol{\Sigma}_c|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_c)\right\}$$

   Use log to handle exponent

   $$\log(L) \propto \left(-\frac{N_c}{2}\log(|\boldsymbol{\Sigma}_c|)\right) + \left(-\frac{1}{2}\sum_n^{N_c}(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_c)\right)$$

Take derivative w.r.t. $\boldsymbol{\mu}_c$

$$\frac{\partial \log(L)}{\partial \boldsymbol{\mu}_c} = -\frac{1}{2}\sum_n^{N_c} 2\boldsymbol{\Sigma}_c^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_c)$$
$$= -\sum_n^{N_c} \boldsymbol{\Sigma}_c^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_c) = 0$$
$$\sum_n^{N_c} \boldsymbol{\Sigma}_c^{-1}\boldsymbol{\mu}_c - \sum_n^{N_c} \boldsymbol{\Sigma}_c^{-1}\mathbf{x}_n = 0$$
$$\boldsymbol{\Sigma}_c^{-1} N_c \boldsymbol{\mu}_c - \boldsymbol{\Sigma}_c^{-1}\sum_n^{N_c} \mathbf{x}_n = 0$$
$$\boldsymbol{\Sigma}_c^{-1} N_c \boldsymbol{\mu}_c = \boldsymbol{\Sigma}_c^{-1}\sum_n^{N_c} \mathbf{x}_n$$
$$N_c \boldsymbol{\mu}_c = \sum_n^{N_c} \mathbf{x}_n$$
$$\boldsymbol{\mu}_c = \frac{1}{N_c}\sum_n^{N} \mathbf{x}_n$$

Take derivative w.r.t. $\boldsymbol{\Sigma}_c$, set to zero, solve

$$\frac{\partial \log(L)}{\partial \boldsymbol{\Sigma}_c} = -\frac{N_c}{2}(\boldsymbol{\Sigma}_c^{-1}) + \frac{1}{2}\sum_n^{N_c}(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-2}(\mathbf{x}_n - \boldsymbol{\mu}_c) = 0$$
$$\frac{N_c}{2}({\boldsymbol{\Sigma}_c}^{-1}) = \frac{1}{2}\sum_n^{N_c}(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-2}(\mathbf{x}_n - \boldsymbol{\mu}_c)$$
$$N_c({\boldsymbol{\Sigma}_c}^{-1}) = \boldsymbol{\Sigma}_c^{-1}\left(\sum_n^{N_c}(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top(\mathbf{x}_n - \boldsymbol{\mu}_c)\right)\boldsymbol{\Sigma}_c^{-1}$$
$$\cancel{N_c({\boldsymbol{\Sigma}_c}^{-1})(\boldsymbol{\Sigma}_c^{-1})} = \boldsymbol{\Sigma}_c^{-1}\left(\sum_n^{N_c}(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top(\mathbf{x}_n - \boldsymbol{\mu}_c)\right)\cancel{\boldsymbol{\Sigma}_c^{-1}(\boldsymbol{\Sigma}_c^{-1})}$$
$$\cancel{\frac{1}{N_c}}\cancel{N_c}({\boldsymbol{\Sigma}_c}^{-1}) = \frac{1}{N_c}\cancel{({\boldsymbol{\Sigma}_c}^{-1})\boldsymbol{\Sigma}_c^{-1}}\left(\sum_n^{N_c}(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top(\mathbf{x}_n - \boldsymbol{\mu}_c)\right)$$
$${\boldsymbol{\Sigma}_c}^{-1} = \frac{1}{N_c}\left(\sum_n^{N_c}(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top(\mathbf{x}_n - \boldsymbol{\mu}_c)\right)$$

2. [4 points] Adapted from **Exercise 5.4** of FCMA p.204:

   Compute the maximum likelihood estimates of q_{mc} for class c of a Bayesian classifier with multinomial class-conditionals and a set of N_c, M-dimensional objects belonging to class c: $\mathbf{x}_1, ..., \mathbf{x}_{N_c}$.

   Solution.

$$P(\mathbf{x}_n|\mathbf{q}) = \left(\frac{s_n!}{\prod_{m=1}^{M} x_{nm}!}\right) \prod_{m=1}^{M} q_m^{x_{nm}}$$

$$L(\mathbf{x}|\mathbf{q}) \propto \prod_{n=1}^{N_c} \prod_{m=1}^{M} {q_{cm}}^{x_{nm}}$$

$$l = \log(L) = \sum_{n=1}^{N} \sum_{m=1}^{M} x_{nm} \log(q_{cm})$$

Need to include constraint, Lagrange multiplier

$$\sum_{m=1}^{M} q_{cm} = 1$$

$$l' = \sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm} \log(q_{cm}) + \lambda(1 - \sum_{m=1}^{M} q_{cm})$$

Take the derivative, set to 0

$$\frac{\partial l'}{\partial q_{cm}} = \frac{\sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm}}{\sum_{m=1}^{M} q_{cm}} - \lambda = 0$$

$$\lambda = \frac{\sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm}}{\sum_{m=1}^{M} q_{cm}}$$

$$\sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm} = \lambda \sum_{m=1}^{M} q_{cm}$$

$$\sum_{m=1}^{M} q_{cm} = \frac{\sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm}}{\lambda}$$

$$\frac{\partial \log(l')}{\partial \lambda} = 1 - \sum_{m=1}^{M} q_{cm} = 0$$

$$\sum_{m=1}^{M} q_{cm} = 1$$

Use the equation three up from here to get:

$$\frac{\sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm}}{\lambda} = 1$$

Solve for λ :

$$\lambda = \sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm}$$

Plug lambda back into this (from above):

$$\sum_{m=1}^{M} q_{cm} = \frac{\sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm}}{\lambda}$$

To get this:

$$\sum_{m=1}^{M} q_{cm} = \frac{\sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm}}{\sum_{n=1}^{N_c} \sum_{m=1}^{M} x_{nm}}$$

I worked on this for hours and just cannot see how to get the correct answer. How can you drop the $\sum_M$ on q_{cm}? Where does m' come from? These are mysteries I cannot solve.

3. [4 points] **Required only for Graduates** Adapted from **Exercise 5.5** of FCMA p.204:

   For a Bayesian classifier with multinomial class-conditionals with M-dimensional parameters $\mathbf{q}_c$, compute the posterior Dirichlet for class c when the prior over $\mathbf{q}_c$ is a Dirichlet with constant parameter α and the observations belonging to class c are the N_c observations $\mathbf{x}_1, ..., \mathbf{x}_{N_c}$.

   Solution.

Likelihood (multinomial):

$$P(\mathbf{x}_n, \mathbf{q}_c) = \left(\frac{s_n!}{\prod_{m=1}^{M} x_{nm}!} \right) \prod_{m=1}^{M} q_{mc}^{x_{nm}}$$

$$P(\mathbf{x}, \mathbf{q}_c) \propto \prod_{n=1}^{N_c} \prod_{m=1}^{M} q_{mc}^{x_{nm}}$$

$$= \prod_{m=1}^{M} q_{mc}^{\sum_{n=1}^{N_c} x_{nm}}$$

Prior (Dirichlet):

$$p(\mathbf{q}_c|\alpha) = \frac{\left(\Gamma \sum_{m=1}^{M} \alpha\right)}{\prod_{m=1}^{M} \Gamma(\alpha)} \prod_{m=1}^{M} q_{mc}^{\alpha-1}$$

Posterior = Likelihood × Prior:

$$\propto \left(\prod_{m=1}^{M} q_{mc}^{\sum_{n=1}^{N_c} x_{nm}} \right) \times \left(\prod_{m=1}^{M} q_{mc}^{\alpha-1} \right)$$

$$= \prod_{m=1}^{M} q_{mc}^{\sum_{n=1}^{N_c} x_{nm} + \alpha - 1}$$

To account for each dimension in M, create a vector:

$$\hat{\alpha} = \left(\sum_{n=1}^{N_c} x_{n1} + \alpha, \sum_{n=1}^{N_c} x_{n2} + \alpha, \cdots \sum_{n=1}^{N_c} x_{nM} + \alpha \right)$$

4. [3 points] For a support vector machine, if we remove one of the support vectors from the training set, does the size of the maximum margin decrease, stay the same, or increase for that dataset? Why? Also justify your answer by providing a simple dataset (no more than 2-dimensions) in which you identify the support vectors, draw the location of the maximum margin hyperplane, remove one of the support vectors, and draw the location of the resulting maximum margin hyperplane. Drawing this by hand is sufficient.

Solution.

I think by definition the margin must increase if you remove one of the supports from a SVM. As demonstrated by the below examples, the margin is maximized according to the support vectors, and removing one support means the margin is maximized again, necessarily to a larger space. It is not only the width that can change, but, as shown in the second example, the direction/slope of the boundary can be greatly changed by the removal.

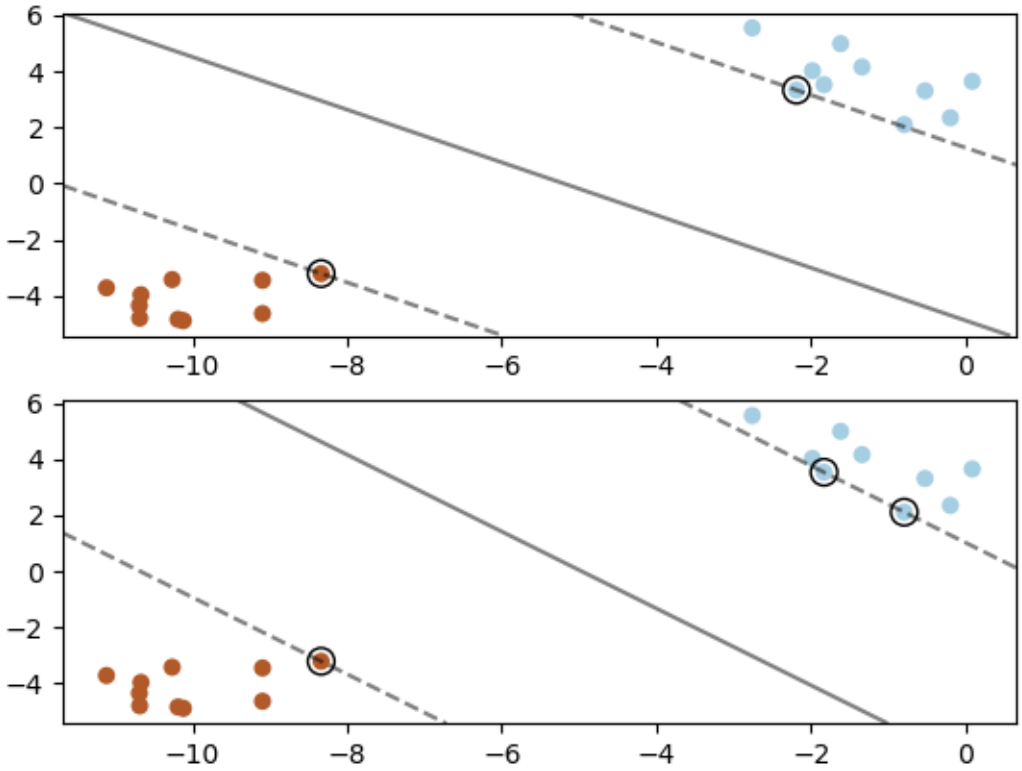

Figure 1: Affect on SVM of removing one support (random seed = 1)

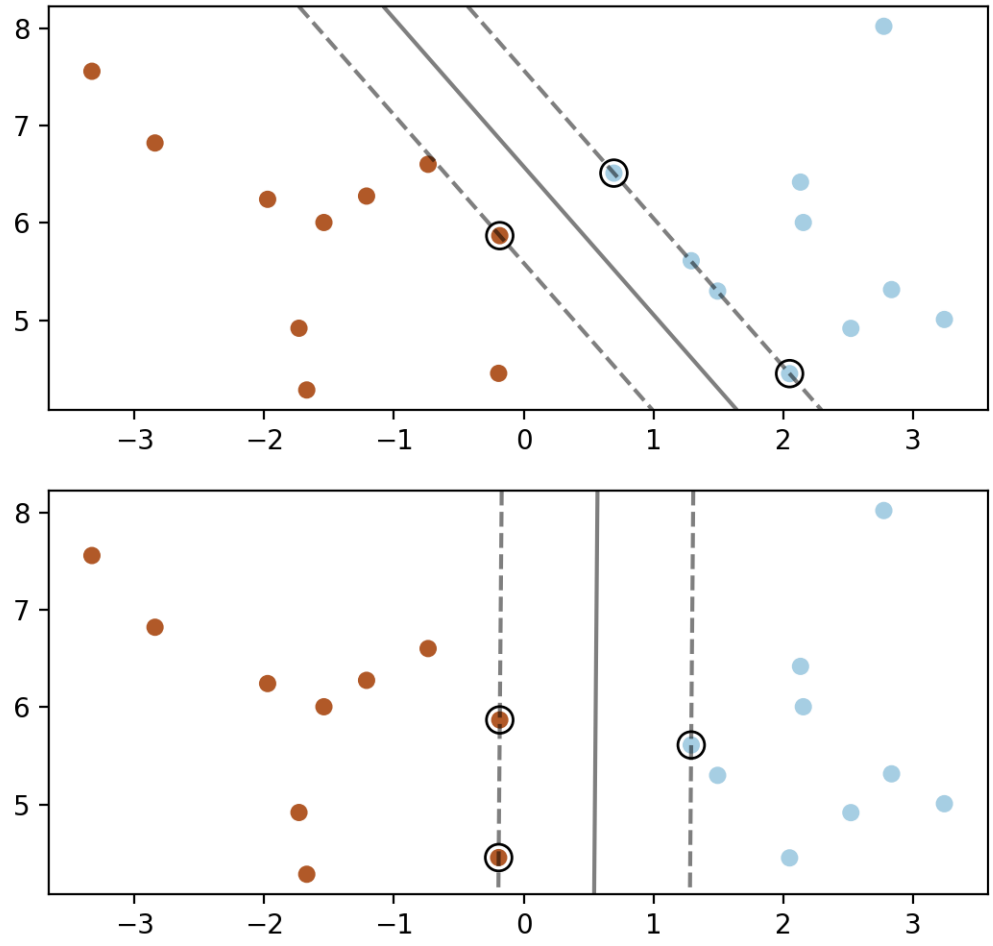

Figure 2: Affect on Support Vector Machine of removing one support (image generated at random, unknown seed)

```
#!/usr/bin/env python3
"""
Author : Ken Youens-Clark <kyclark@email.arizona.edu>
Date   : 2018-12-02
Purpose: Demonstrate affect on SVM of removing a support vector
"""

import argparse
import sys
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.datasets.samples_generator import make_blobs
from sklearn import svm


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Demonstrate affect on SVM of removing a support vector' ,
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument(
        '-o',
        '--outfile',
        help='File to write figure',
        metavar='FILE',
        type=str,
        default=None)

    parser.add_argument(
        '-r',
        '--random_seed',
        help='Random seed value',
        metavar='int',
        type=int,
        default=None)

    return parser.parse_args()


# --------------------------------------------------
def warn(msg):
    """Print a message to STDERR"""
    print(msg, file=sys.stderr)


# --------------------------------------------------
def die(msg='Something bad happened'):
    """warn() and exit with error"""
    warn(msg)
    sys.exit(1)


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()

    fig, ax = plt.subplots(2)

    if args.random_seed is not None:
```

```
    np.random.seed(args.random_seed)

X, y = make_blobs(n_samples=20, centers=2)

clf = svm.SVC(kernel='linear', C=1000)
clf.fit(X, y)

#
# Create grid to evaluate model
#
xlim = ax[0].get_xlim()
ylim = ax[0].get_ylim()
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

#
# Plot data, decision boundary and margins, support vectors
#
ax[0].scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

ax[0].contour(
    XX,
    YY,
    Z,
    colors='k',
    levels=[-1, 0, 1],
    alpha=0.5,
    linestyles=['--', '-', '--'])

ax[0].scatter(
    clf.support_vectors_[:, 0],
    clf.support_vectors_[:, 1],
    s=100,
    linewidth=1,
    facecolors='none',
    edgecolors='k')

#
# Now remove one of the supports and refit
#
support = clf.support_
X2 = np.delete(X, support[0], axis=0)
y2 = np.delete(y, support[0])

clf.fit(X2, y2)

Z2 = clf.decision_function(xy).reshape(XX.shape)
```

```
    ax[1].scatter(X2[:, 0], X2[:, 1], c=y2, s=30, cmap=plt.cm.Paired)

    ax[1].contour(
        XX,
        YY,
        Z2,
        colors='k',
        levels=[-1, 0, 1],
        alpha=0.5,
        linestyles=['--', '-', '--'])

    ax[1].scatter(
        clf.support_vectors_[:, 0],
        clf.support_vectors_[:, 1],
        s=100,
        linewidth=1,
        facecolors='none',
        edgecolors='k')

    if args.outfile:
        warn('Saving figure to "{}"'.format(args.outfile))
        plt.savefig(args.outfile)

    plt.show()


# --------------------------------------------------
if __name__ == '__main__':
    main()
```

5. [4 points] In this exercise you will use the python script, `knn.py`, which contains code to plot the decision boundary for a k-nearest neighbors classifier. Two data sources are also provided in the `data` directory: `knn_binary_data.csv` and `knn_three_class_data.csv`.

   In python file, the function `knn` is to compute the k-nearest neighbors classification for a point `p`, but the functionality is not currently implemented. You *can* run the code in its unimplemented state, but the decision boundary will not display (everything will be considered the same class). You must implement this yourself. Use a Euclidean distance measure for determining the neighbors (note: you don' t need to take the square root! The sum of squares is sufficient). You do not need to implement any fancy indexing – you can simply search for the k nearest points by searching over the pairwise distance of the input (`p`) to all of the "training" inputs (`x`). Use the maximum class frequency as the class decision rule. You do not need to worry about doing anything special for breaking ties. The numpy function `argsort` and the python `collections.Counter` may be of use, but are not required. Submit the `knn.py` file with your implementation. In your written solution, run the code on both of the provided data sources (`knn_binary_data.csv` and `knn_three_class_data.csv`.), and for each, plot the decision boundary for $k = \{1, 5, 10, 59\}$. Include informative captions and describe any patterns you see in the decision boundaries as k changes.

   Solution.

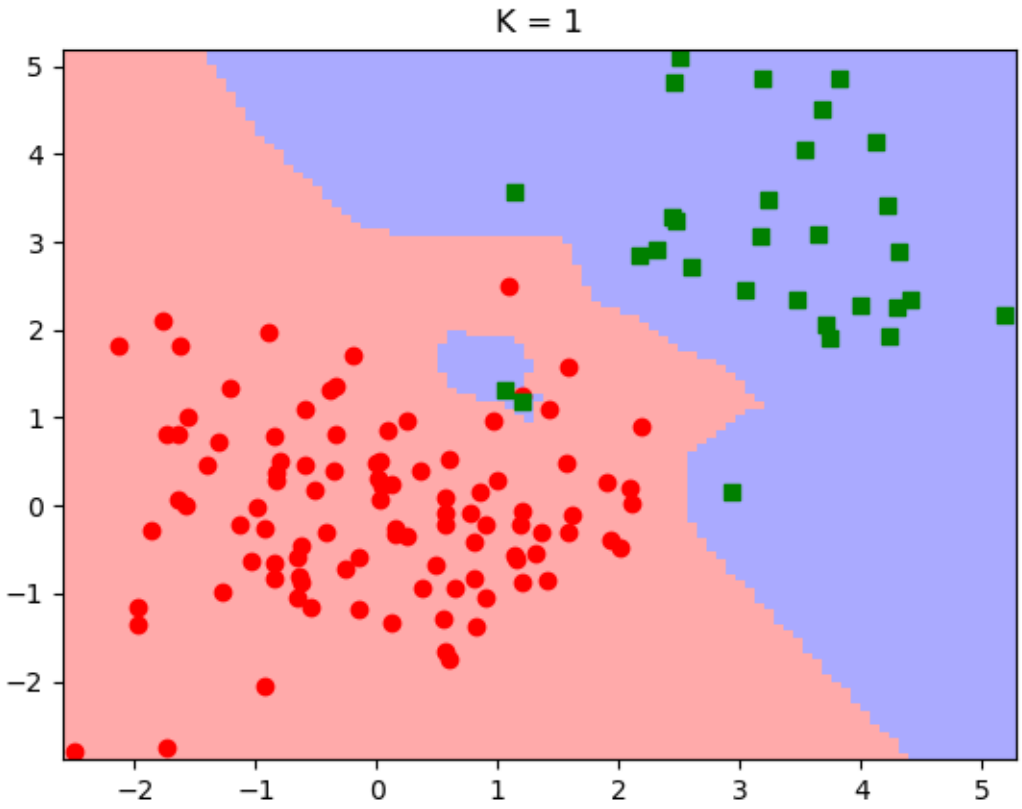

Figure 3: KNN k=1 binary classification. Notice the island of blue inside the red where two green points are clearly in the wrong place. When k=1, the boundaries are too close, creating islands of the green squares when they ought to be assumed to have crossed into enemy territory.

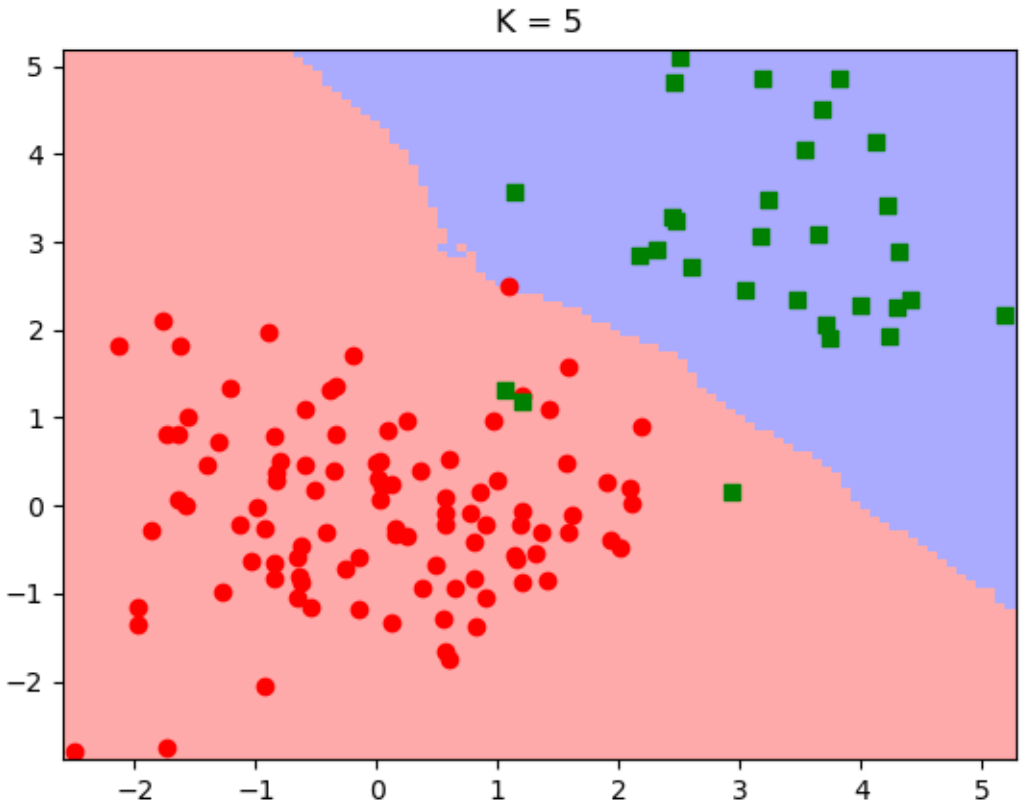

Figure 4: KNN k=5 binary classification. Here the boundary appears much smoother, and the three green squares all appear to be in red circle's territory.

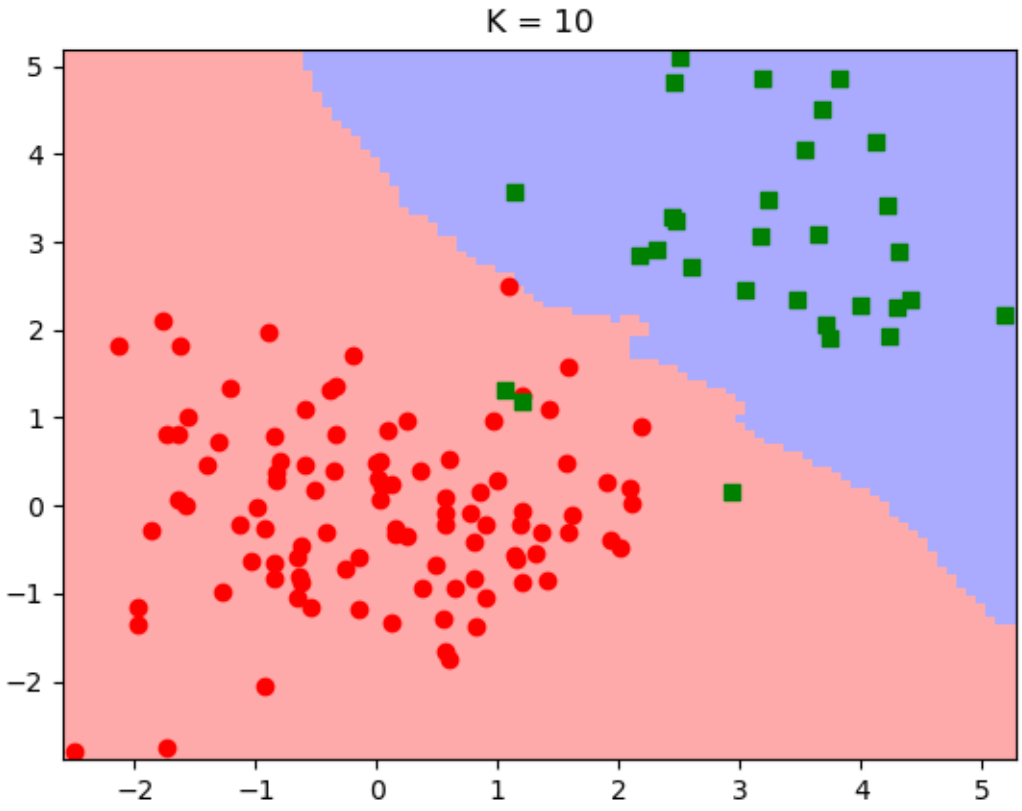

Figure 5: KNN k=10 binary classification. At k=10, now the red territory is pushing further into green's because there are more of them.

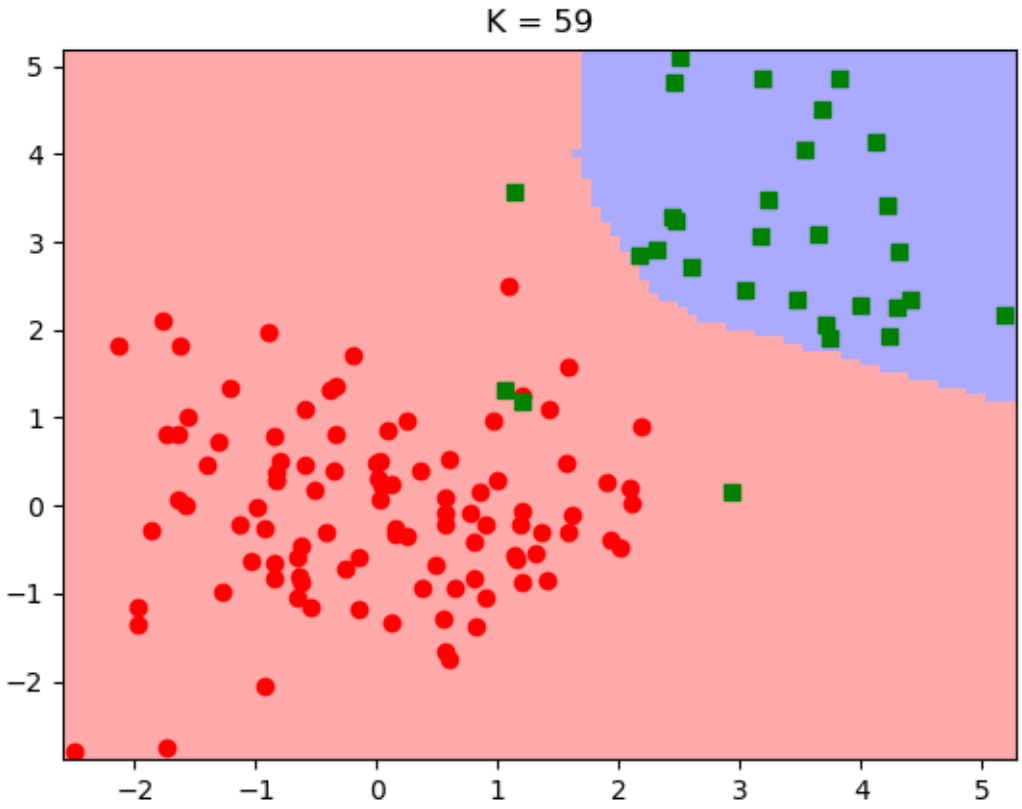

Figure 6: KNN k=59 binary classification. At k=59, red's majority is clearly creating a great deal of influence, pushing green into the corner.

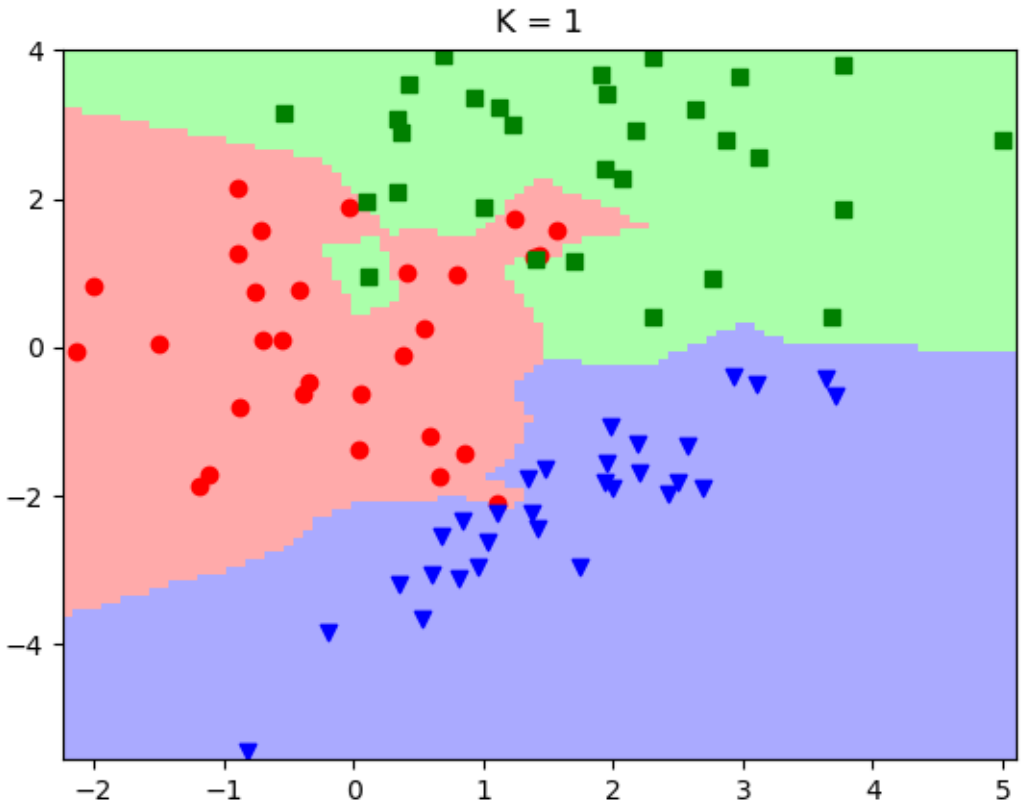

Figure 7: KNN k=1 three class classification. For the classification of three classes, k of 1 again is too liberal with creating boundaries for instance the one green square that is clearly in the red circle's area and also a red circle that ventures into the blue region.

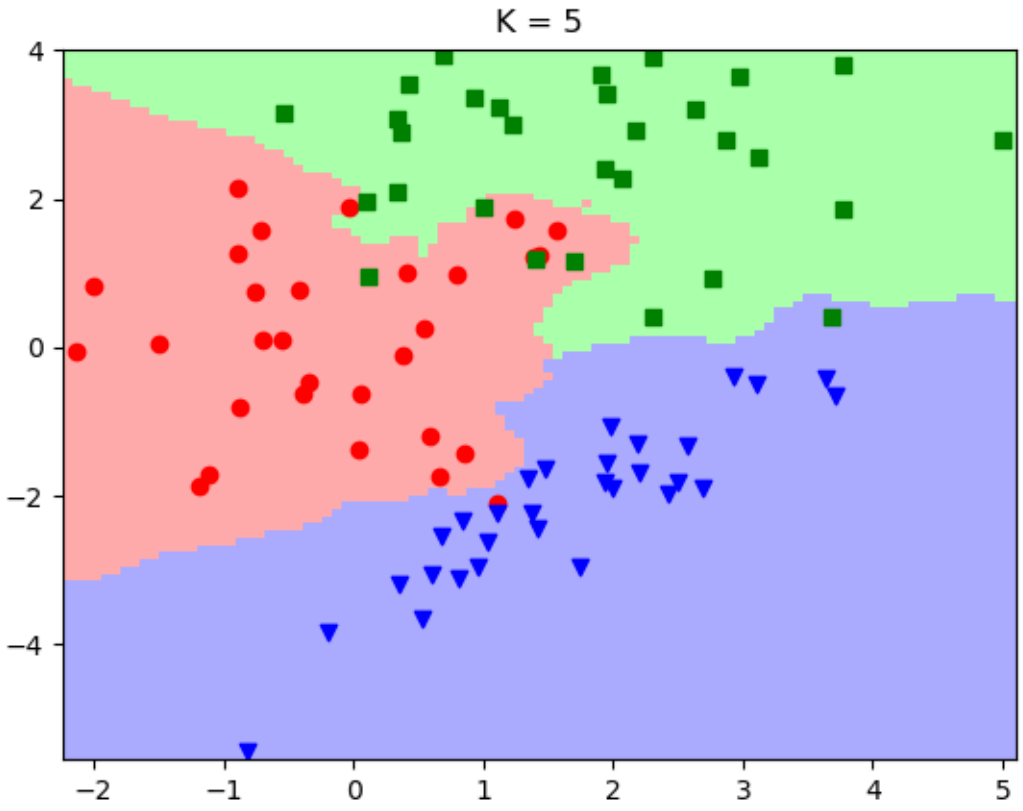

Figure 8: KNN k=5 three class classification. Again k of 5 and 10 create smoother boundaries that allow errant positions to wander where they ought not.

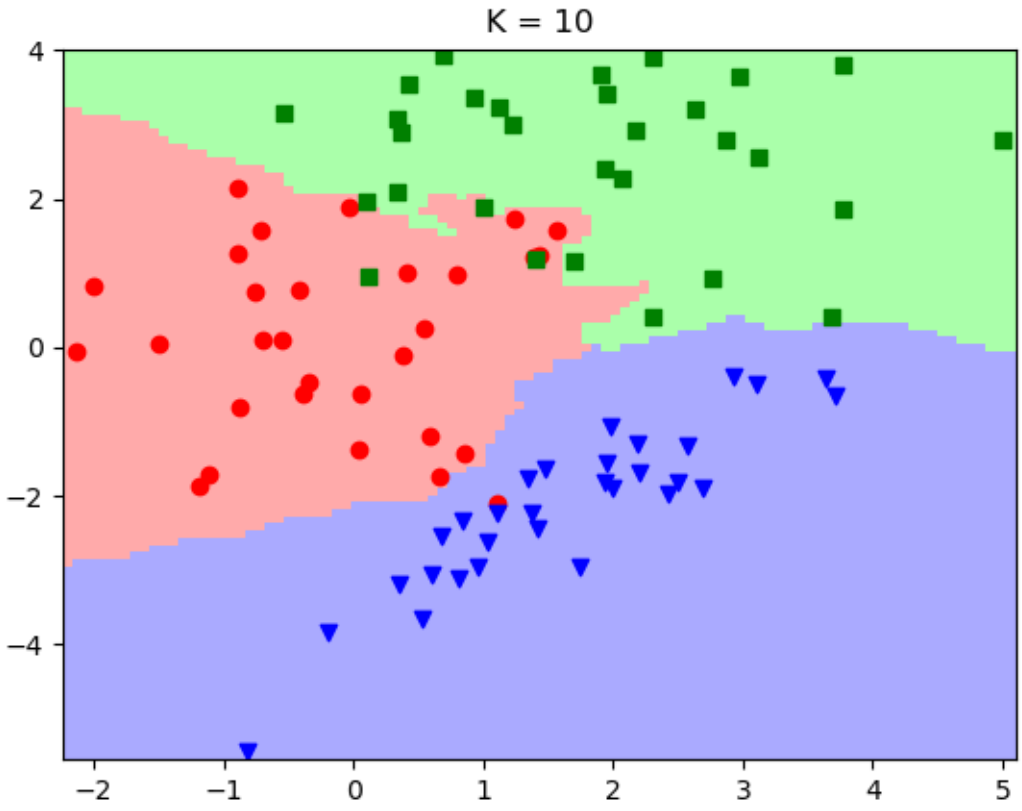

Figure 9: KNN k=10 three class classification. K of 10 does about as well at 5.

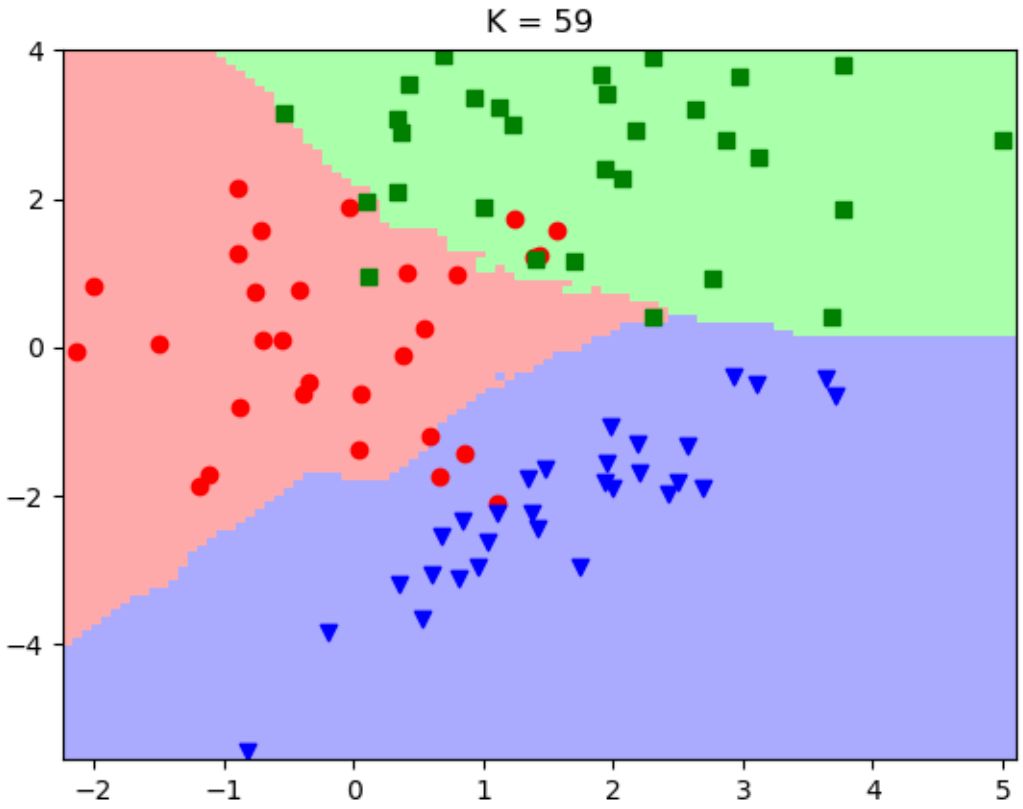

Figure 10: KNN k=59 three class classification. K of 59 isn't actually terrible since there are roughly equal numbers of the three classes and so no one class gets marginalized too much, but it does a poorer job than 5 or 10.

```
#!/usr/bin/env python3
"""
Author : Ken Youens-Clark <kyclark@email.arizona.edu>
Date   : 2018-11-24
Purpose: K-Nearest Neighbors
"""

import argparse
import matplotlib
import os
#matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import numpy as np
import sys
from collections import Counter
from matplotlib.colors import ListedColormap
from scipy.spatial import distance
```

```
# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='K-Nearest Neighbors',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument(
        '-f',
        '--file',
        metavar='FILE',
        help='Input file',
        default='../data/knn_binary_data.csv')

    parser.add_argument(
        '-k',
        metavar='INT',
        help='Values for K',
        nargs='+',
        type=int,
        default=[1, 5, 10, 59])

    parser.add_argument(
        '-o',
        '--out_dir',
        help='Output directory for saved figures',
        metavar='DIR',
        type=str,
        default=None)

    parser.add_argument(
        '-g',
        '--granularity',
        help='Granularity',
        metavar='int',
        type=int,
        default=100)

    parser.add_argument(
        '-q', '--quiet', help='Do not show figures', action='store_true')

    return parser.parse_args()


# --------------------------------------------------
def warn(msg):
    """Print a message to STDERR"""
    print(msg, file=sys.stderr)
```

```
# ------------------------------------------------
def die(msg='Something bad happened'):
    """warn() and exit with error"""
    warn(msg)
    sys.exit(1)


# ------------------------------------------------
def read_data(path, d=','):
    """
    Read 2-dimensional real-valued features with associated class labels
    :param path: path to csv file
    :param d: delimiter
    :return: x=array of features, t=class labels
    """
    arr = np.genfromtxt(path, delimiter=d, dtype=None)
    length = len(arr)
    x = np.zeros(shape=(length, 2))
    t = np.zeros(length)
    for i, (x1, x2, tv) in enumerate(arr):
        x[i, 0] = x1
        x[i, 1] = x2
        t[i] = int(tv)
    return x, t


# ------------------------------------------------
def knn(p, k, x, t):
    """
    K-Nearest Neighbors classifier.  Return the most frequent class among the k
    nearest points
    :param p: point to classify (assumes 2-dimensional)
    :param k: number of nearest neighbors
    :param x: array of observed 2-dimensional points
    :param t: array of target labels (corresponding to points)
    :return: the top class label
    """

    d = np.argsort(list(map(lambda z: distance.euclidean(p, z), x)))[:k]
    count = Counter(t[d])

    # most_common() returns a sorted list of tuples
    # so take the first element of the first tuple -- [0][0]
    return count.most_common(1)[0][0]


# ------------------------------------------------
def plot_decision_boundary(k, x, t, granularity=100, out_file=None):
    """
    Given data (observed x and labels t) and choice k of nearest neighbors,
    plots the decision boundary based on a grid of classifications over the
```

```
feature space.
:param k: number of nearest neighbors
:param x: array of observed 2-dimensional points
:param t: array of target labels (corresponding to points)
:param granularity: controls granularity of the meshgrid
:return:
"""
print('KNN for K={0}'.format(k))

# Initialize meshgrid to be used to store the class prediction values
# this is used for computing and plotting the decision boundary contour
Xv, Yv = np.meshgrid(
    np.linspace(np.min(x[:, 0]) - 0.1,
                np.max(x[:, 0]) + 0.1, granularity),
    np.linspace(np.min(x[:, 1]) - 0.1,
                np.max(x[:, 1]) + 0.1, granularity))

# Calculate KNN classification for every point in meshgrid
classes = np.zeros(shape=(Xv.shape[0], Xv.shape[1]))
for i in range(Xv.shape[0]):
    for j in range(Xv.shape[1]):
        classes[i][j] = knn(np.array([Xv[i][j], Yv[i][j]]), k, x, t)

# plot the binary decision boundary contour
plt.figure()

# Create color map
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
plt.pcolormesh(Xv, Yv, classes, cmap=cmap_light)
ti = 'K = {0}'.format(k)
plt.title(ti)
plt.draw()

# Plot the points
ma = ['o', 's', 'v']
fc = ['r', 'g', 'b']  # np.array([0, 0, 0]), np.array([1, 1, 1])]
tv = np.unique(t.flatten())  # an array of the unique class labels

#if new_figure:
#    plt.figure()

for i in range(tv.shape[0]):
    # returns a boolean vector mask for selecting just the instances of class tv[i]
    pos = (t == tv[i]).nonzero()
    plt.scatter(
        np.asarray(x[pos, 0]),
        np.asarray(x[pos, 1]),
        marker=ma[i],
        facecolor=fc[i])

if out_file:
```

```
        warn('Saving figure to "{}"'.format(out_file))
        plt.savefig(out_file)


# ------------------------------------------------
def main():
    args = get_args()
    in_file = args.file
    K = args.k
    granularity = args.granularity
    out_dir = args.out_dir

    if not os.path.isfile(in_file):
        die('"{}" is not a file'.format(in_file))

    x, t = read_data(in_file)

    basename, _ = os.path.splitext(os.path.basename(in_file))

    if out_dir:
        out_dir = os.path.abspath(out_dir)

    # Loop over different neighborhood values K
    for k in K:
        out_file = None
        if out_dir:
            out_file = os.path.join(out_dir, '{}-k-{}.png'.format(basename, k))
        plot_decision_boundary(
            k, x, t, granularity=granularity, out_file=out_file)

    if not args.quiet:
        warn('Showing figures')
        plt.show()

    warn('Done')


# ------------------------------------------------
if __name__ == '__main__':
    main()
```

6. [4 points] Using your implementation of your KNN classifier in exercise 5, write a script to perform 10-fold cross-validation in a search for the best choice of K. Remember to randomize your data at the start of the CV procedure, but use the same CV folds for each K. Make your script search in the range of $1 \leq K \leq 30$. Run your script on both data sources: `knn_binary_data.csv` and `knn_three_class_data.csv`. In your written solution, provide a plot of K (x-axis) to classification error (ratio of points misclassified to total; y-axis) for each data set, and report the *best* k for each. Submit your script as a python file named `knn-cv`.

   Solution.

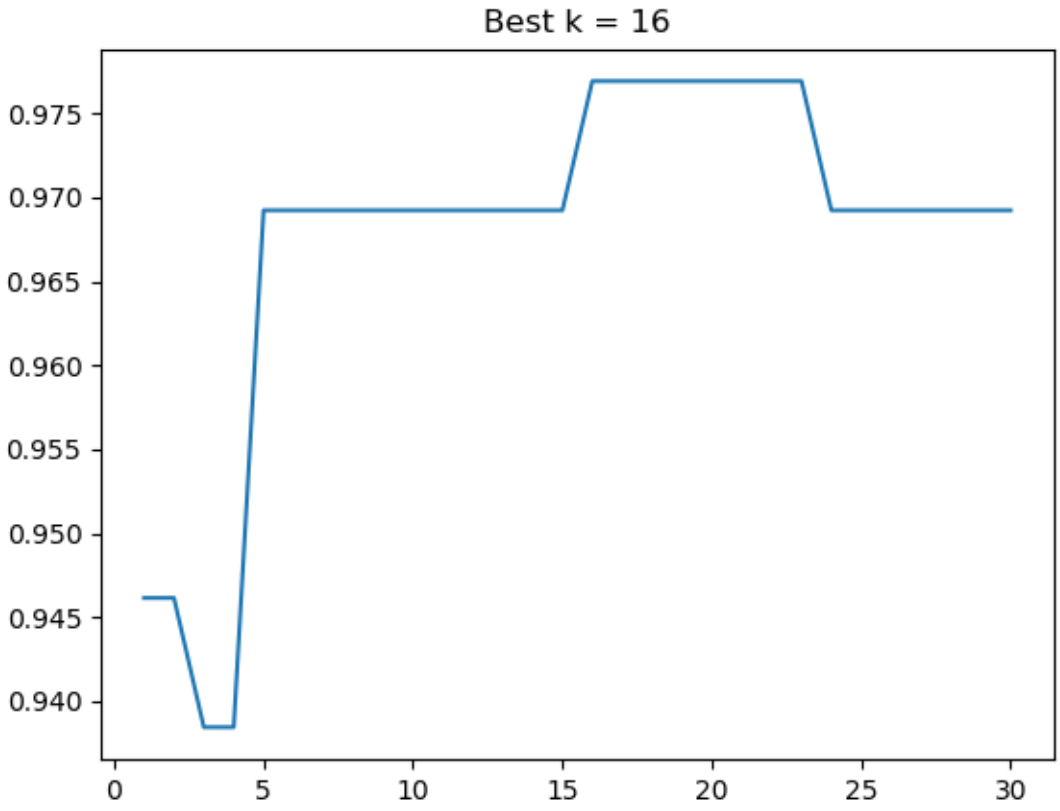

Figure 11: 10-fold CV of KNN on binary classification where $1 \leq k \leq 30$

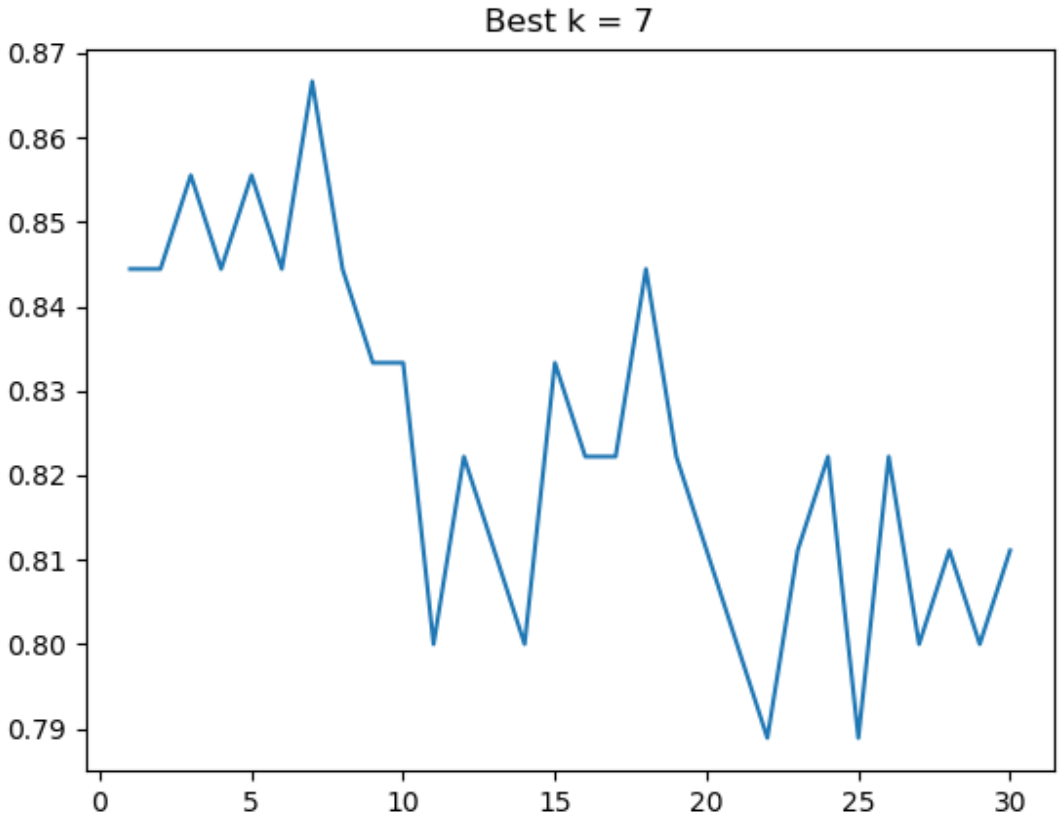

Figure 12: 10-fold CV of KNN on three-class classification where $1 \leq k \leq 30$

```
#!/usr/bin/env python3
"""
Author : Ken Youens-Clark <kyclark@email.arizona.edu>
Date   : 2018-11-24
Purpose: Cross-validation of K-Nearest Neighbors
"""

import argparse
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from scipy.spatial import distance
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split


# --------------------------------------------------
```

```
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Cross-validation of K-Nearest Neighbors',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument(
        '-f',
        '--file',
        help='Input file',
        metavar='FILE',
        type=str,
        default='../data/knn_binary_data.csv')

    parser.add_argument(
        '-i',
        '--iterations',
        help='How many iterartions for CV',
        metavar='int',
        type=int,
        default=10)

    parser.add_argument(
        '-o',
        '--outfile',
        help='File name to write figure',
        metavar='FILE',
        type=str,
        default='')

    parser.add_argument(
        '-k', help='Max value for K', metavar='int', type=int, default=30)

    parser.add_argument(
        '-q', '--quiet', help='Do not show figures', action='store_true')

    return parser.parse_args()


# --------------------------------------------------
def warn(msg):
    """Print a message to STDERR"""
    print(msg, file=sys.stderr)


# --------------------------------------------------
def die(msg='Something bad happened'):
    """warn() and exit with error"""
    warn(msg)
    sys.exit(1)
```

```
# --------------------------------------------------
def knn(p, k, x, t):
    """
    K-Nearest Neighbors classifier.  Return the most frequent class among the k
    nearest points
    :param p: point to classify (assumes 2-dimensional)
    :param k: number of nearest neighbors
    :param x: array of observed 2-dimensional points
    :param t: array of target labels (corresponding to points)
    :return: the top class label
    """

    d = np.argsort(list(map(lambda z: distance.euclidean(p, z), x)))[:k]
    count = Counter(t[d])
    return count.most_common(1)[0][0]


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    infile = args.file
    K = args.k
    iterations = args.iterations
    out_file = args.outfile

    def debug(msg):
        if not args.quiet:
            warn(msg)

    df = pd.read_csv(infile, header=None)
    X = df.iloc[:, 0:2].values
    t = pd.to_numeric(df.iloc[:, 2].values, downcast='integer')
    splits = list(KFold(n_splits=iterations).split(X))
    x_plot = []
    y_plot = []

    for k in range(1, K + 1):
        predictions = []
        for (train, test) in splits:
            predicted = list(
                map(lambda p: knn(p, k, X[train], t[train]), X[test]))
            predictions.append(np.mean(predicted == t[test]))

        average = np.mean(predictions)
        debug('k {} = {}'.format(k, average))
        x_plot.append(k)
        y_plot.append(average)

    best = np.argmax(y_plot) + 1
```

```
    plt.plot(x_plot, y_plot)
    plt.title('Best k = {}'.format(best))

    if out_file:
        debug('Saving figure to "{}"'.format(out_file))
        plt.savefig(out_file)

    if not args.quiet:
        debug('Showing figure')
        plt.show()

    debug('Done')


# ------------------------------------------------
if __name__ == '__main__':
    main()
```