ML 2016/17
Exercise 3: K-NN

07/11/16

General information

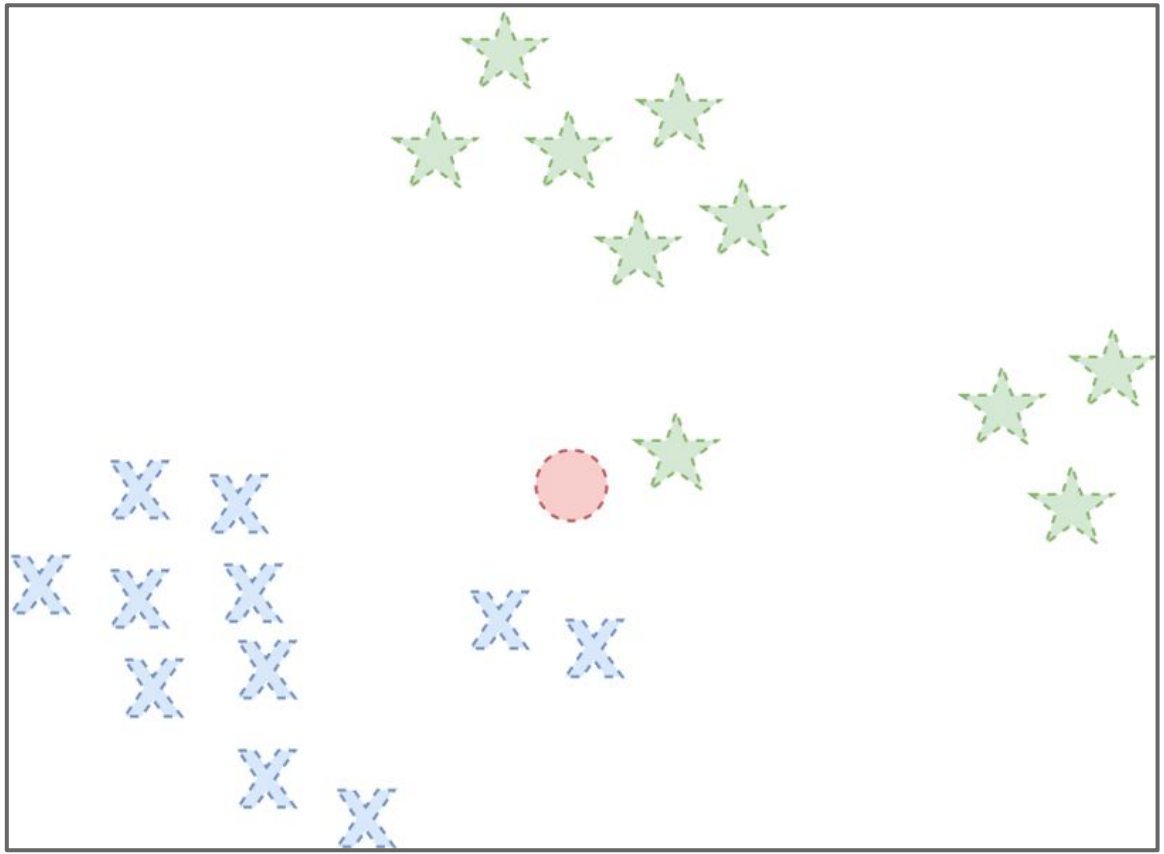
- The assignments are not graded on a scale: it's simply pass/no pass
  - If one homework is not sufficient you can simply redo it
- All assignments must be delivered one month before **you** take the exam
- Submission through email: send to fabiom.carlucci@dis.uniroma1.it
- Questions can be written to same email address.
- Office hours to meet in person: **Wednesday** at B004 (Via Ariosto, the door in front of library), 10AM-12PM.
- Python recommended: https://www.continuum.io/downloads
- *There is no need to replicate exactly the images I show!*

HW2: K-NN

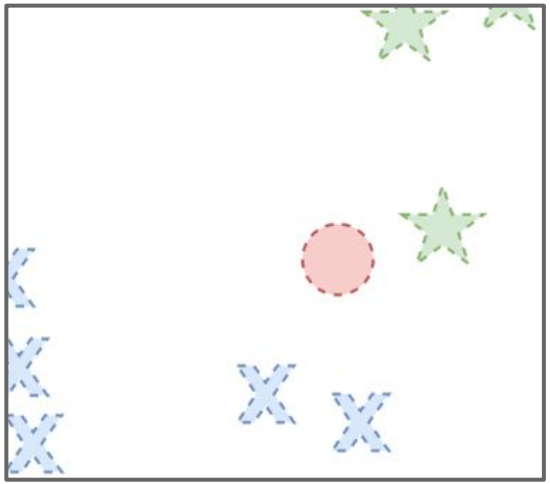
- Familiarize with the basic concepts of K Nearest Neighbors
- Explore different weight functions
- Implement NN in Python

Once you complete the experience send the report to fabiom.carlucci@dis.uniroma1.it with subject **“[ML1617] KNN report”**

How does classification work in KNN?



KNN in practice

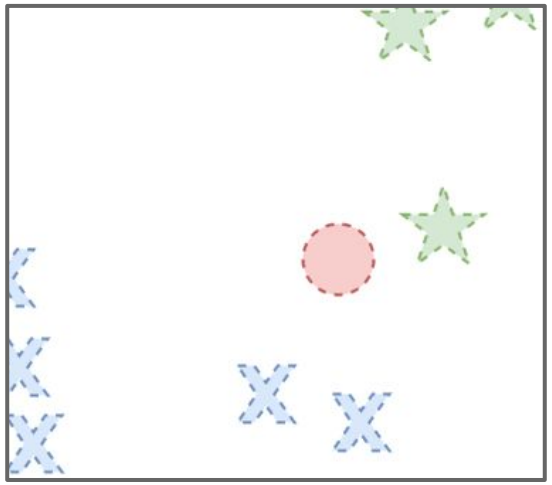


k is the number of closest points which vote to assign a class label.

For ***k = 1*** we will look only at the closest point. In this case the query point will be assigned the **green star** class.

For ***k = 3*** the blue points are also considered and are in majority. Thus the query point will be assigned the **blue cross** class.

KNN in practice



For ***k = 2*** the nearest neighbours are now 1 **green star** and 1 **blue cross**. Which class will be assigned?

It depends on the **weight function**.

If the neighbors are unweighted, each of of the NNs will contribute equally, for ***k=2*** this is a draw.

We can also decide to weight the neighbors based on their distance, in that case the query points will be assigned the ***green star*** class.

Many different weight functions are possible!

What to do 1/2

1. Load ***Iris*** dataset
2. Apply PCA to reduce to 2 dimensions (remember to standardize the data)
3. Randomly split data into train and test splits. Test split should be 40% of the whole data
4. For ***k*** from 1 to 10:
    a. apply nearest neighbour search
    b. report accuracy, plot data and decision boundaries
5. How do the boundaries change? Why?
6. Select ***k=3*** and show how the decision boundary changes when using different weight functions (read the description in the docs). Try:
    a. ‘uniform’
    b. ‘distance’

What to do 2/2

7. Write your own weight function to compute the gaussian function of the square of the distance: $w = e^{-\alpha d^2}$
8. Plot decision boundaries for $\alpha = [0.1, 10, 100, 1000]$
9. How do the boundaries change? Why?
10. Do a grid search to find the optimal number of neighbours and the best weight function for this data (try ‘uniform’, ‘distance’ and your gaussian function)
11. Plot decision boundary of best solution and discuss it
12. (Optional) Write your own implementation of Nearest Neighbors using only Numpy. To make this simple, consider only the closest neighbour (we don’t need to bother with weight functions, why?) of a given point. Compare accuracy with sklearn’s implementation

Step by step 1

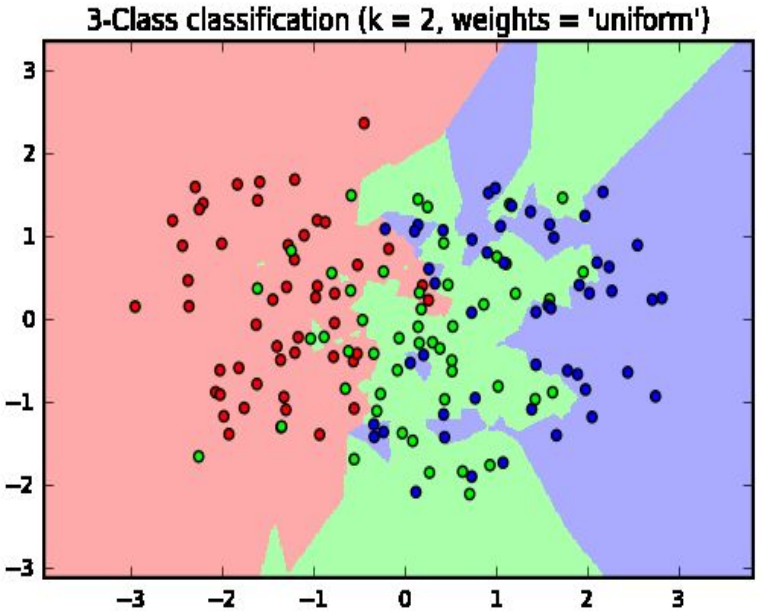
Load Iris:

```
from sklearn import neighbors, datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

Then standardize, apply PCA and get your train, test splits (check past experiences)

Fit and test KNN (http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html):

```
clf = neighbors.KNeighborsClassifier(n_neighbors)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

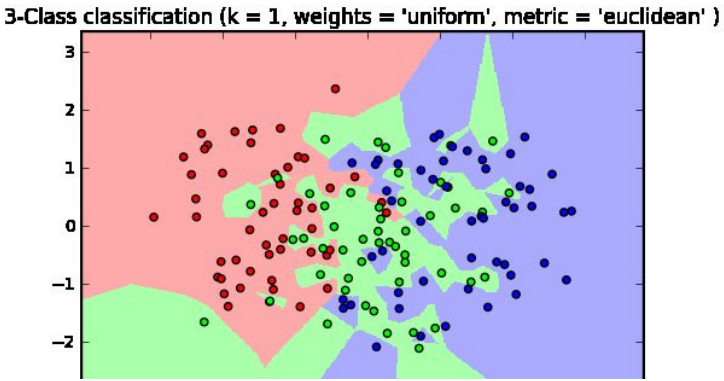


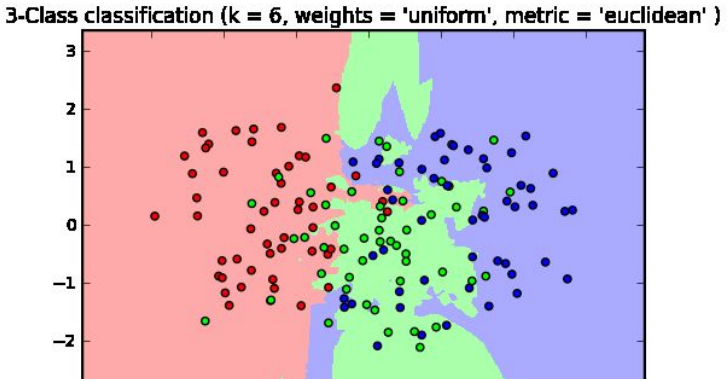
Step by step 2

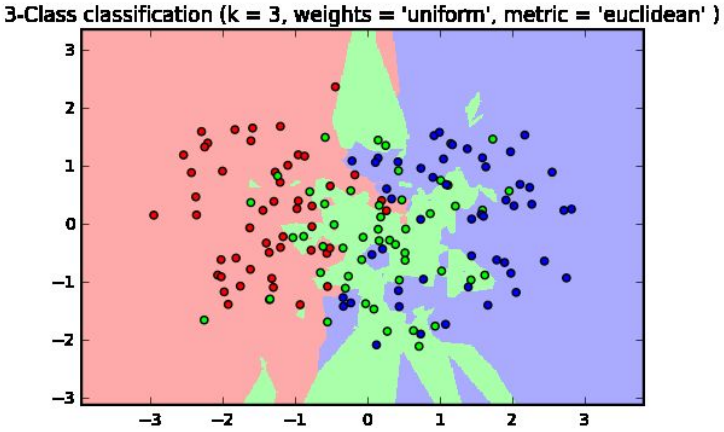
How do you plot a decision boundary?

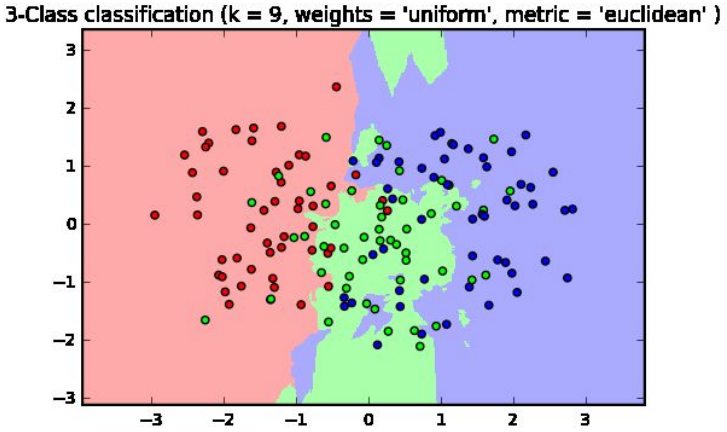
```
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

Step by step 3 - changing k



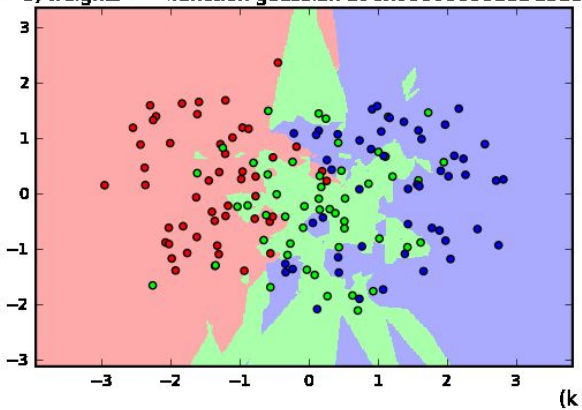




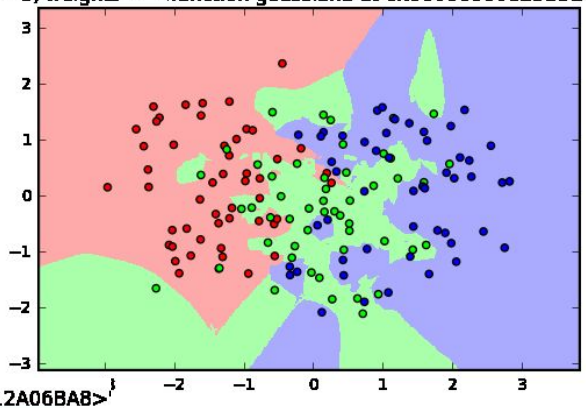


Step by step 4 - weight functions

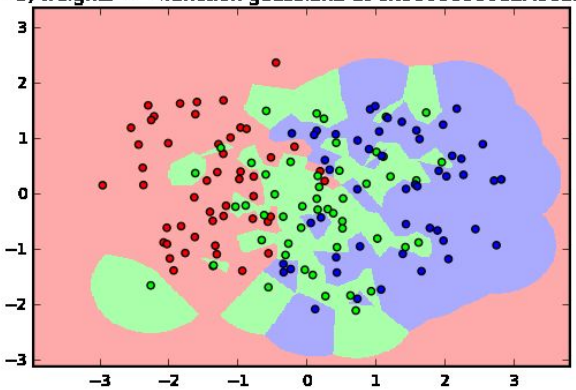
k = 3, weights = '<function gaussian at 0x0000000011DC3B38>



(k = 3, weights = '<function gaussian1 at 0x0000000125B3E48>',



(k = 3, weights = '<function gaussian2 at 0x0000000012A06BA8>'



Your turn now! Questions?