# Homework Report 3

Manuel Del Verme
1769408

December 15, 2016

## 1 Data preparation

The data set consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray. The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.

### 1.1 Standardization

PCA (2 dimensions), centering and scaling were applied to the data

### 1.2 Train, test split

40% of the dataset was used to test the classification

## 2 K Nearest Neighbors

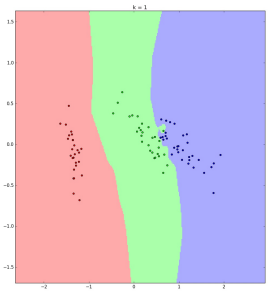
K-NN classifies a new point as the class which appears more frequently in the k-nearest points seen during the training, we can also use a weight function to consider points more or less relevant based on the weight function value.

### 2.1 values for K

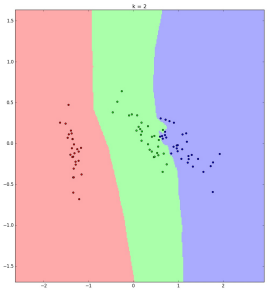
We tried values of k in [1, 10]

| k | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| accuracy | 0.95 | 0.9 | 0.93 | 0.93 | 0.95 | 0.95 | 0.95 | 0.97 | 0.95 | 0.97 |

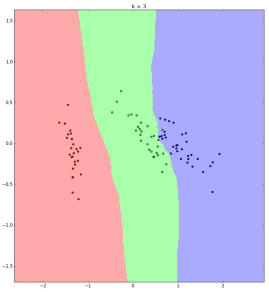
Looking at the error trends and decision boundaries in Figure 1, very little improves with higher values of k, this could be caused by an high signal to noise ratio, or a test set which is too small, I would say the latter since the accuracy varied up to 10% between runs.



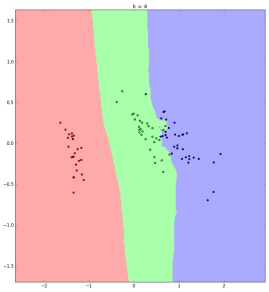
(a) K = 1



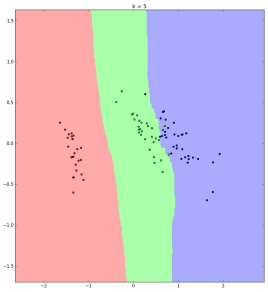
(b) K = 2



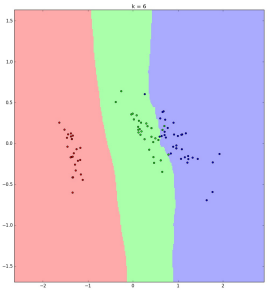
(c) K = 3



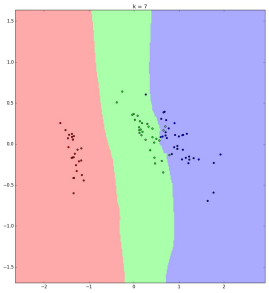
(d) K = 4



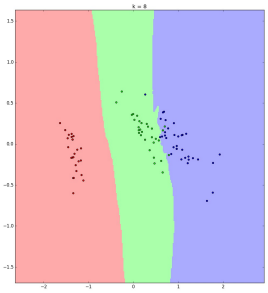
(e) K = 5



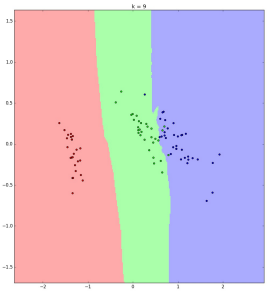
(f) K = 6



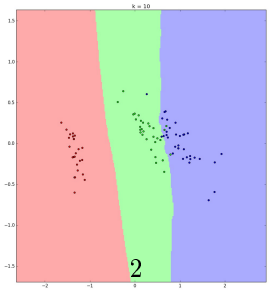
(g) K = 7



(h) K = 8

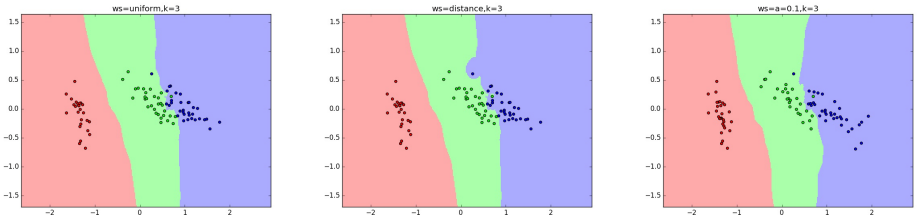


(i) K = 9



(j) K = 10

Figure 1: various values of K




(a) uniform (b) distance (c) $\alpha$=0.1

(d) $\alpha$=10 (e) $\alpha$=100 (f) $\alpha$=1000

Figure 2: Various $\alpha$

## 2.2 weight functions

using k=3, some weight functions were tested

| weight function | uniform | distance | $e^{-0.1d^2}$ | $e^{-10d^2}$ | $e^{-100d^2}$ | $e^{-1000d^2}$ |
|---|---|---|---|---|---|---|
| accuracy | 0.93 | 0.95 | 0.93 | 0.93 | 0.95 | 0.95 |

Again the accuracy doesn't really improve the classification.
Figure 2 shows boundaries for the weight functions, the strange boundary for $\alpha = 1000$ could be because of gating during the distance calculations or gating caused by numerical error, numerical error seem to be at fault here since there was no mention of thresholds in the documentation nor in the (python) source of sklearn.

## 2.3 hyper-parameter selection

the highest score was a draw by several combinations:

| k | 4 | 4 | 4 | 6 | 6 | 6 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| weight | distance | $\alpha$=0.1 | $\alpha$=10 | distance | $\alpha$=0.1 | $\alpha$=10 | $\alpha$=100 | $\alpha$=100 |

## 2.4 implemented K-NN

Figure 4 shows a simple implementation which could easily be expanded to include weights (by multiplying the distances variable) and K neighbors with

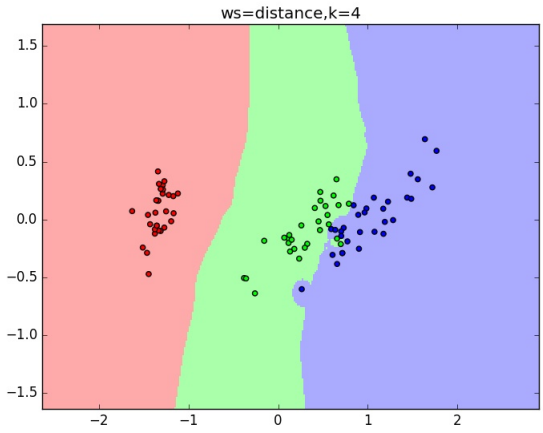



Figure 3: one of the highest scoring best combinations

```
def fit(self, Xs, Ys):
    self.Xs = np.tile(Xs, Xs.shape[0])
    self.Ys = Ys
def predict(self, x):
    x_prime = np.zeros((self.Xs.shape[0], x.shape[1]))
    x_prime[:x.shape[0], :x.shape[1]] = x
    b = x_prime.reshape((1, -1)).repeat(self.Xs.shape[0], axis=0)
    dist_map = self.Xs - b
    dist_map = np.power(dist_map, 2)
    classes = []
    last = x.shape[0] * x.shape[1]
    for i in range(0, last, x.shape[1]):
        extractor = np.zeros(self.Xs.shape[0] * x.shape[1])
        extractor[i:i + x.shape[1]] = 1
        distances = dist_map.dot(extractor)
        classes.append(self.Ys[np.argmin(distances)])
    return classes
```

Figure 4: Python implementation

np.argsort() . The algorithm does not approximate any distance so this algorithm doesn't lose any optimality.

### 2.4.1 weight function

as long as K=1 the weight function would not change the results since with there is no comparison between neighbors.