Sheet10

June 28, 2022

Exercise 20

a) The main idea of the k-NN algorithm is to say, that an arbitrary value x_0 will have a similar value as its k neighbours. When dealing with greatly differing attributes, the neighbours will be further away, the prediction will only consider values that are in a similar magnitude. The model will have increased bias. To solve this problem, the best idea is to reduce the distance betweenn the attributes, while still mantaining the relative distances. One of the ways to do this, as seen in the last sheet, is to apply the 10-logarithm, improving the scaling between the attributes.

b) It is called a lazy learner, as there are no discrete learning processes involved, such as gradient descent or improving of the information gain. The predictor function is fixed for each point after finding the k-nearest neighbours. To find the nearest neighbours of the point x_0 we will need to iterate over all other N-1 points in the dataset. Since this process is repeated until we reach x_n (we have n data points), the runtime will have $O(n^2)$. As the only difference between learning and application is the point whose neighbours are checked, the runtime will also be $O(n^2)$. -> Training and application take the same time!

Other algorithms usually tend to have a longer training runtime and fast application.

For a random forest with a depth of d and t trees, in each node all data points need to be considered. Since the algorithm is greedy and only decides the best split for each node, the processes in each node are independant from each other, resulting in $O(2^d \cdot n \cdot t)$. d and t are much smaller than n thus we have $O(n)$

The application process to n datapoints will take $O(n \cdot d \cdot t)$, which also becomes $O(n)$ since d and t are much smaller than the number of data points.

(c) Implement a -NN algorithm for the classification of events. Follow the class structure given in the attached file class_structure.py. The method predict should output a numpy array containing the predicted label for each sample. Procedure: For each event to be classified:

1) Calculation of distances to all points of the training sample.

2) Determine the training events with the smallest distance (note: determine only the indices of the events instead of sorting the array itself). Hint: The Numpy function numpy.argsort() can be useful

[ ]:
```
def fit(self, X, y):
        '''Fit routine.
        Training data is stored within object.
```

```
        Parameters
        ----------
        X : numpy.array, shape=(n_samples, n_attributes)
            Training data.
        y : numpy.array shape=(n_samples)
            Training labels.
        '''
        # Code
        y = np.array(y)

        self.X_train = X

        distances = spatial.distance.cdist(X, X, 'euclidean')

        distances_indeces = np.argsort(distances)

        k_closest = distances_indeces[:, 1:self.k+1]

        y_closest = [y[k_closest[i]] for i in range(len(X))]


        predictions = [np.sum(y_closest[i])/self.k > 0.5 for i in␣
 ↪range(len(y_closest))]

        self.predictions_training = np.array(predictions)



def predict(self, X):
    '''Prediction routine.
    Predict class association of each sample of X.

    Parameters
    ----------
    X : numpy.array, shape=(n_samples, n_attributes)
        Data to classify.

    Returns
    -------
    prediction : numpy.array, shape=(n_samples)
        Predictions, containing the predicted label of each sample.
    '''
    # Code

    distances = spatial.distance.cdist(X, self.X_train, 'euclidean')

    distances_indeces = np.argsort(distances)
```

```
    k_closest = distances_indeces[:, 1:self.k+1]

    y_closest = [self.predictions_training[k_closest[i]] for i in range(len(X))]

    predictions = [np.sum(y_closest[i])/self.k > 0.5 for i in␣
↪range(len(y_closest))]

    return predictions
```

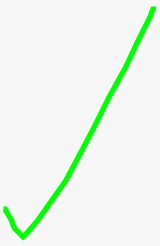
d) Apply your algorithm to the neutrino Monte Carlo of sheet 5. Use the NeutrinoMC.hdf5 file provided in Moodle.

- Use the attributes CountHits, x and y.
- Set = 10.
- Use 5000 events as a training set.
- The test set shall consist of 20 000 underground and 10 000 signal events.

Determine recall, precision and significance.

[1]:
```
import class_structure
import numpy as np
import pandas as pd

f1 = pd.HDFStore('NeutrinoMC.hdf5', mode = 'r')


df_Background = f1.get('/Background')
df_Background = df_Background.dropna()
df_Signal = f1.get('/Signal')

#instance = class_structure.KNN(10)
df_Background['Label'] = np.full(len(df_Background['NumberOfHits']), False)

df_Signal['Label'] = np.full(len(df_Signal['NumberOfHits']), True)

del df_Signal["Energy"]
del df_Signal["AcceptanceMask"]

df_Signal = df_Signal.dropna()

df_Signal_training = df_Signal.sample(n = 2500)
df_Background_training= df_Background.sample(n= 2500)

frames = [df_Background_training, df_Signal_training]

training = pd.concat(frames)
```

[2]:
```
classifier = class_structure.KNN(k = 10)

X = training[["NumberOfHits", "x", "y"]]
X_training = np.array(X)
y = training['Label']

classifier.fit(X_training,y)

test_Background = df_Background.sample(n = 20000)
test_Signal = df_Signal.sample(n = 10000)

test_set = pd.concat([test_Background, test_Signal])

y_test = test_set['Label']
del test_set['Label']

predictions = np.array(classifier.predict(test_set))

y_test = np.array(y_test)

def recall(y, predictions):

    TP_mask = np.logical_and(predictions == y, predictions == True)
    TP = np.sum(TP_mask)

    FN_mask = np.logical_and(predictions != y, predictions == False)
    FN = np.sum(FN_mask)

    return TP/(TP+FN)

def precision(y, predictions):
    TP_mask = np.logical_and(predictions == y, predictions == True)
    TP = np.sum(TP_mask)

    FP_mask = np.logical_and(predictions != y, predictions == True)
    FP = np.sum(FP_mask)

    return TP/(TP+FP)

def significance(y, predictions):

    # keine klare Definition in der VL dazu gefunden

    return None
```



```
print('Recall', recall(y_test, predictions))
print('Precision', precision(y_test, predictions))
```

```
Recall 0.9593
Precision 0.7668878407546567
```

(e) What changes if you use log10(CountHits) instead of CountHits?

[3]:
```
hits = X['NumberOfHits']
X['NumberOfHits'] = np.log10(hits)

classifier.fit(X,y)

testhits = np.array(test_set['NumberOfHits'])
test_set['NumberOfHits'] = np.log10(testhits)


predictions = np.array(classifier.predict(test_set))

y_test = np.array(y_test)

print('Recall', recall(y_test, predictions))
print('Precision', precision(y_test, predictions))
```

```
/tmp/ipykernel_69535/1684855474.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['NumberOfHits'] = np.log10(hits)
```

```
Recall 0.9847
Precision 0.8404028334898012
```

Recall and Precision is improved, the error described in (a) is reduced as the attributes are now closer to eachother -> improved clustering

(f) What changes if you use $= 20$ instead of $= 10$?

[4]:
```
classifier = class_structure.KNN(k = 20)

X = training[["NumberOfHits", "x", "y"]]
y = training['Label']

classifier.fit(X,y)

test_Background = df_Background.sample(n = 20000)
```

```
test_Signal = df_Signal.sample(n = 10000)

test_set = pd.concat([test_Background, test_Signal])

y_test = test_set['Label']
del test_set['Label']

predictions = np.array(classifier.predict(test_set))

y_test = np.array(y_test)

print('Recall', recall(y_test, predictions))
print('Precision', precision(y_test, predictions))
```

```
Recall 0.9549
Precision 0.7270443124714482
```

The recall as well as the precision are worse now. More neighbours increase the probability of overfitting -> worse performance on test set

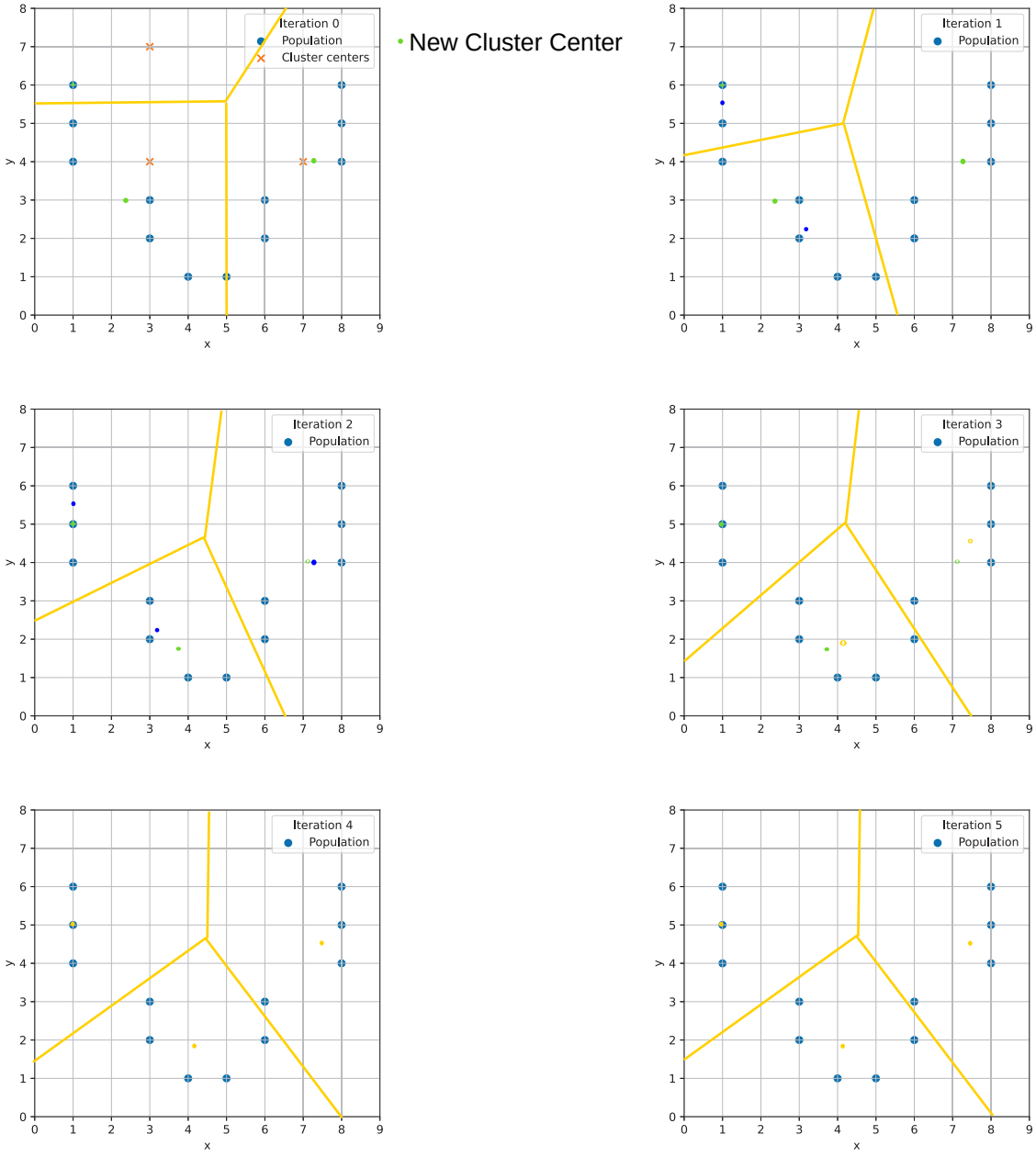


Figure 1: Populations and templates to draw the cluster centers and cluster boundaries for task 21