PRACTICE 2

Data Mining

Marc Llort Maulion

marc.llort

Marc Llort Maulion (marc.llort)

Goal

The main objectives of this practice are:

- Learn to master Python package scikit-learn
- Take fluency with the selection, standardization, preprocessing and search attributes.
- Knowing the different implementations of the algorithm-based learning with examples IBL.
- Working with the concept of optimizing a learner.

To perform all these functions, you will need the following previously imported libraries:

```
import scipy
from scipy import stats
import numpy
import sklearn
import sklearn.datasets
from sklearn.model_selection import KFold
from sklearn.decomposition import KernelPCA
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
import sklearn.metrics
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import matplotlib.pyplot as pyplot
```

Marc Llort Maulion (marc.llort)

Analyze the data set of digits (Digits data set)

At first, once imported various bookstores, touch us load *datasets* bookstore "skelearn":

```
# Carregem datasets

digits = sklearn.datasets.load_digits()

X = digits.data
Y = digits.target
```

After uploading datasets, that we will have a variable X a matrix of 1797 rows by 64 columns while Y have stored a matrix of 1797 rows by a column.

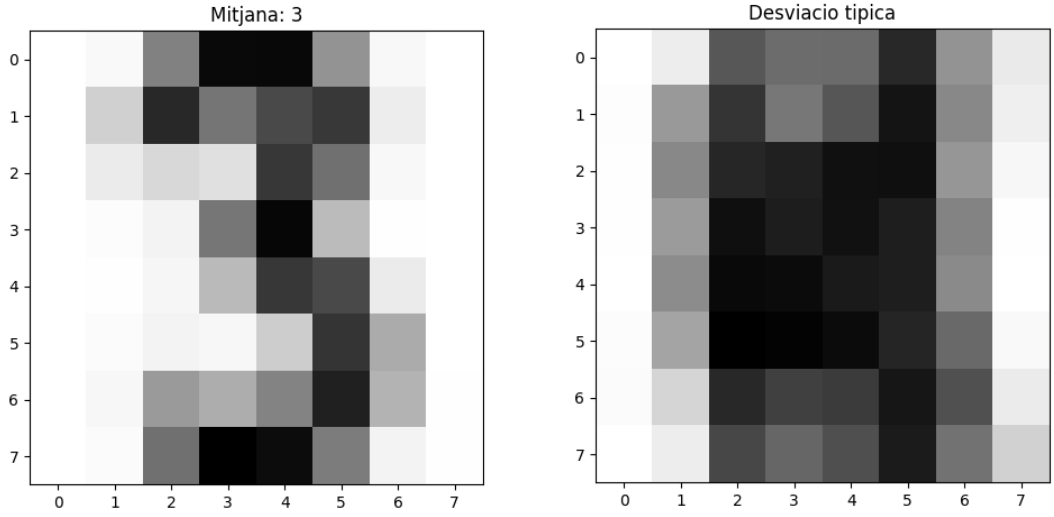
To calculate the mean, standard deviation and the number of elements of training for each class: (Please note that all calculations are made will result in the value of the third test)

```
# 1. Analitzar el conjunt de dades de digits (Digits Data Set)
valorTest = 3

# Numero de mostres per classe
unique, counts = numpy.unique(Y, return_counts=True)
print(dict(zip(unique, counts)))

# Calcul de la mitjana
mitjana = numpy.mean(X[Y == valorTest, :], axis=0)
print("Mitjana ", valorTest, ": ", mitjana)

pyplot.imshow(mitjana.reshape(8, 8), interpolation="none", cmap="Greys")
pyplot.title("Mitjana: " + str(valorTest))
pyplot.show()

# Calcul de la desviacio tipica
std = numpy.std(X, axis=0)
pyplot.imshow(std.reshape(8, 8), interpolation="none", cmap="Greys")
pyplot.title("Desviacio tipica")
pyplot.show()
```

Marc Llort Maulion (marc.llort)

```
>>> runfile('/Users/marcllort/PycharmProjects/MineriaP2/mineriaP2.py', wdir='/Users/marcllort/PycharmProjects/MineriaP2')
{0: 178, 1: 182, 2: 177, 3: 183, 4: 181, 5: 182, 6: 181, 7: 179, 8: 174, 9: 180}
('Mitjana ', array([0.00000000e+00, 6.44808743e-01, 8.38797814e+00, 1.41693989e+01,
       1.42240437e+01, 7.48087432e+00, 7.86885246e-01, 5.46448087e-03,
       1.09289617e-02, 4.20765027e+00, 1.26557377e+01, 8.98907104e+00,
       1.12841530e+01, 1.19945355e+01, 2.10928962e+00, 1.63934426e-02,
       5.46448087e-03, 2.22404372e+00, 3.70491803e+00, 3.10928962e+00,
       1.20273224e+01, 9.32240437e+00, 8.14207650e-01, 0.00000000e+00,
       0.00000000e+00, 2.95081967e-01, 1.45901639e+00, 8.93989071e+00,
       1.42732240e+01, 5.60655738e+00, 8.19672131e-02, 0.00000000e+00,
       0.00000000e+00, 6.01092896e-02, 1.04918033e+00, 5.63934426e+00,
       1.20491803e+01, 1.12786885e+01, 2.19672131e+00, 0.00000000e+00,
       0.00000000e+00, 4.26229508e-01, 1.39890710e+00, 9.67213115e-01,
       4.40983607e+00, 1.21366120e+01, 6.31693989e+00, 0.00000000e+00,
       0.00000000e+00, 8.68852459e-01, 7.11475410e+00, 6.22950820e+00,
       8.26229508e+00, 1.30163934e+01, 5.92896175e+00, 6.55737705e-02,
       0.00000000e+00, 5.02732240e-01, 9.31693989e+00, 1.46502732e+01,
       1.39726776e+01, 8.67213115e+00, 1.40983607e+00, 6.55737705e-02]))
('Desviacio tipica: ', array([0.        , 0.90693964, 4.75350317, 4.24765948, 4.28619491,
       5.66484088, 3.32484969, 1.03709417, 0.09419533, 3.19527098,
       5.41994694, 3.97643575, 4.78134964, 6.05127561, 3.58532293,
       0.82768465, 0.06235094, 3.57530605, 5.68918332, 5.80104695,
       6.17400993, 6.19559718, 3.25896254, 0.43847543, 0.03334258,
       3.14565685, 6.19031469, 5.88129939, 6.15038083, 5.87092136,
       3.68543009, 0.04712725, 0.        , 3.4794038 , 6.32292731,
       6.26664682, 5.93183902, 5.86901393, 3.53629836, 0.        ,
       0.14514503, 2.98098645, 6.53613529, 6.43958504, 6.25776954,
       5.69394162, 4.32974601, 0.30727036, 0.20416633, 1.74566694,
       5.64292531, 5.22549314, 5.30057302, 6.02947606, 4.91803706,
       0.98412698, 0.02358333, 0.9340418 , 5.1015993 , 4.37347662,
       4.93257433, 5.89898069, 4.08940957, 1.85960409]))
```

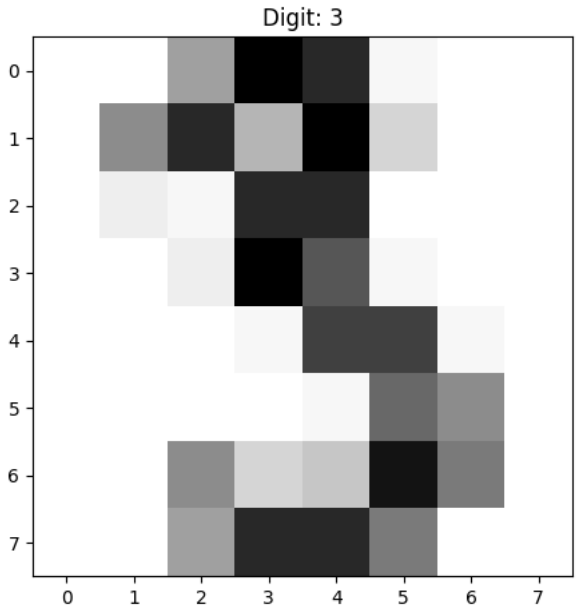
The resulting plots and the average deviation respectively:



To implement the Optional simply we need, when we imported the library perform the following: (value remains the third test)

```
# Opcional: imshow de valorTest
pyplot.imshow(X[valorTest].reshape(8, 8), interpolation="none", cmap="Greys")
pyplot.title("Digit: " + str(Y[valorTest]))
pyplot.show()
```

And the result is the following plot:



Division to train and test and normalization of data

In this section we divide the variables train between 70% and 30% test. Once divided we normalize the data X (train and test) in order to ensure that they are focused to 0 and have a standard deviation of one.

To carry out this section you will function:

train_test_split (X, Y, test_size = 00:30, train_size = 0.70)

This function is divided into sub-arrays and random test train.

The return that provides this function is:

Split: list length = 2 * len (matrices)

where *split* is a list containing the division test train tickets. Once we have the data necessary to normalize them:

```
# 2. Divisio amb train i test i normalitzacio de les dades

X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_split(X, Y, test_size=0.30, train_size=0.70)
X_train_norm = scipy.stats.zscore(X_train, axis=1, ddof=1)
X_test_norm = scipy.stats.zscore(X_test, axis=1, ddof=1)
```

Marc Llort Maulion (marc.llort)

Screening in different major components

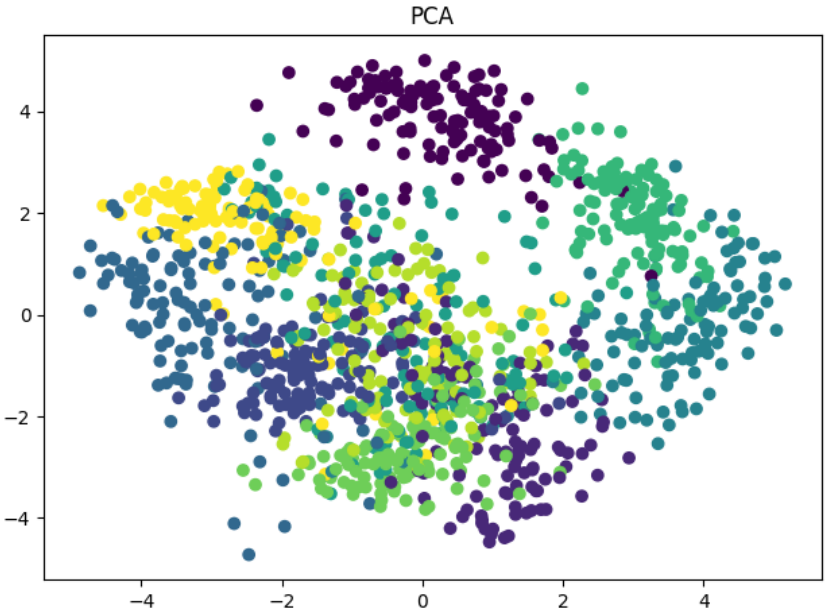
In this section you will sklearn.decomposition.X functions, where X and PCA will TruncatedSVD.

To make the **principal component analysis (PCA)** will:

```
# Descomposem les dades amb PCA
pca = PCA(n_components=2).fit_transform(X_train_norm)
pyplot.scatter(pca[:, 0], pca[:, 1], c=Y_train)
pyplot.title("PCA")
pyplot.show()
```

the **PCA** for reducing the dimensionality of linear singular value decomposition of the data to project to a lower dimensional space.

Using Scatterplot we would:

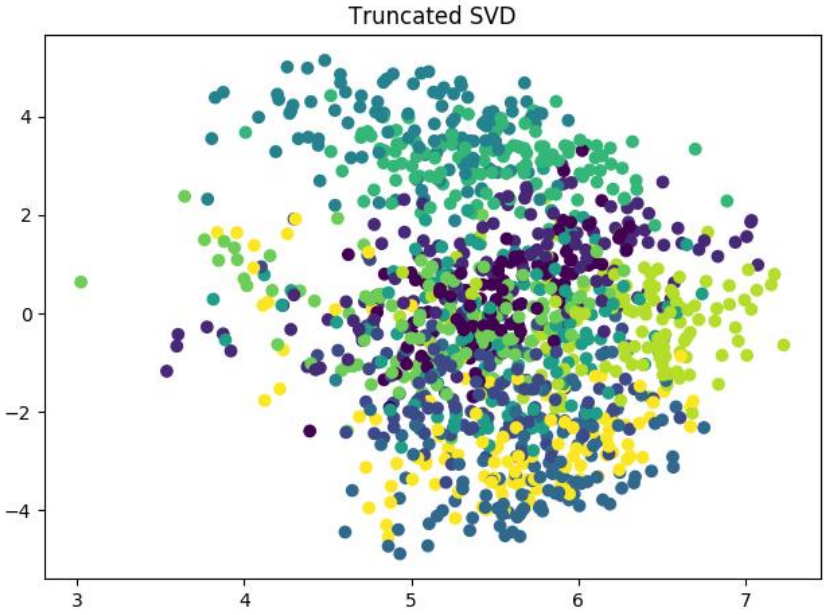


To make a **singular value decomposition (SVD)** will:

```
# Descomposem les dades amb Truncated SVD
svd = TruncatedSVD(n_components=2).fit_transform(X_train_norm)
pyplot.scatter(svd[:, 0], svd[:, 1], c=Y_train)
pyplot.title("Truncated SVD")
pyplot.show()
```

the **SVD** making a linear reduction of dimensionality by truncated singular value decomposition (SVD). Unlike PCA, the data estimator centered before computing the singular value decomposition. This means you can work with arrays *scipi.sparse* efficiently.

Using Scatterplot we would:

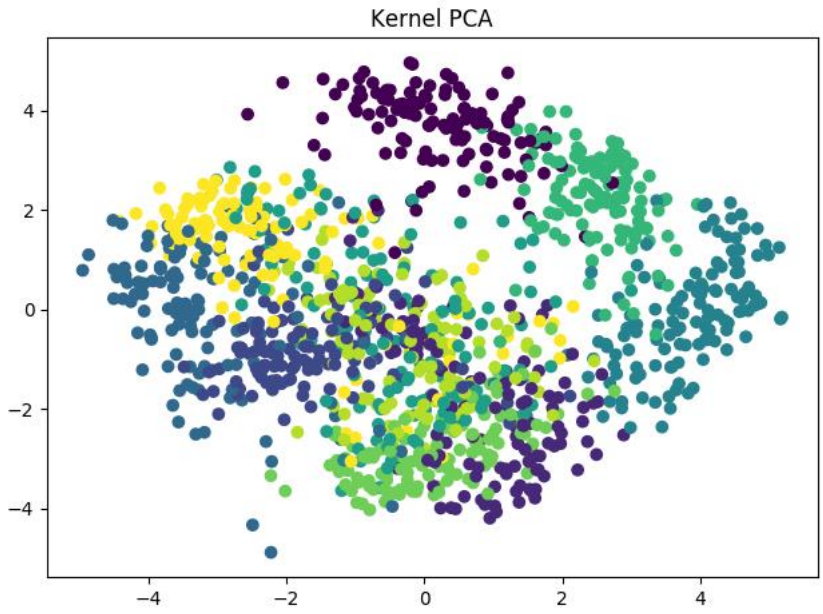


Finally, another option for a breakdown and achieve better separation of data can do for you **KernelPCA:**

```
# Descomposem les dades amb kernel PCA
kernelpca = KernelPCA(n_components=2).fit_transform(X_train_norm)
pyplot.scatter(kernelpca[:, 0], kernelpca[:, 1], c=Y_train)
pyplot.title("Kernel PCA")
pyplot.show()
```

the **KernelPCA** for a reduced dimensionality by using nonlinear kernels.

Using Scatterplot we would:



Marc Llort Maulion (marc.llort)

Use cross-validation to estimate the optimal number of neighbors K

In this section you will function:

sklearn.model_selection.KFold

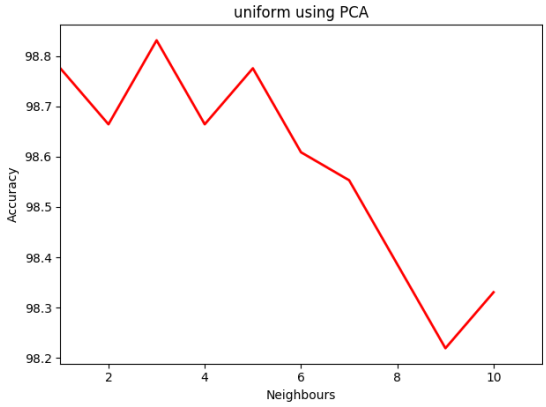
to a division in 10-fold cross validation to estimate the optimal number of first neighbors, the optimal number of dimensions.

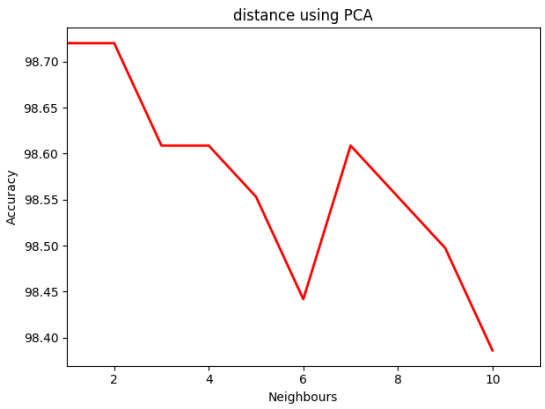
First, we define our function test, where we have combined the two functions for values crossvalidats according to certain parameters.

```
for i in range(min_neighbours, max_neighbours):
    values.append(i)

def get_folds(n_splits):
    kfold = KFold(n_splits=n_splits, shuffle=True)
    folds = list(kfold.split(X))
    return folds


def get_predicted(train_X, train_Y, test_X, n_neighbours, weight):
    knearest = KNeighborsClassifier(n_neighbors=n_neighbours, weights=weight)
    model = knearest.fit(train_X, train_Y)
    predicted_y = model.predict(test_X)
    return predicted_y


def compute_test(n_splits, min_neighbours, max_neighbours, weight):
    # Veins i dimensions optimes
    folds = get_folds(n_splits)
    scores = []

    # Proves amb numero de veins variant
    for i in range(min_neighbours, max_neighbours):
        success = 0
        total = 0

        for j in range(len(folds)):
            train, test = folds[j]

            x_train = [X[k] for k in train]
            x_test = [X[k] for k in test]
            y_train = [Y[k] for k in train]
            y_test = [Y[k] for k in test]

            # Prediccio en base al classificador
            predicted_y = get_predicted(x_train, y_train, x_test, i, weight)

            for k in range(len(y_test)):
                total = total + 1
                if y_test[k] == predicted_y[k]:
                    success = success + 1

        scores.append(float(success) / (float(total)) * 100)
    return scores


scores = compute_test(n_splits, min_neighbours, max_neighbours, weight)

# Resultats
fig = pyplot.figure()
ax = fig.add_subplot(111)
ax.plot(values, scores, color='red', linewidth=2)
ax.set_xlim(min_neighbours, max_neighbours)
pyplot.title(weight + " using SVD")
pyplot.xlabel("Neighbours")
pyplot.ylabel("Accuracy")
pyplot.show()
```

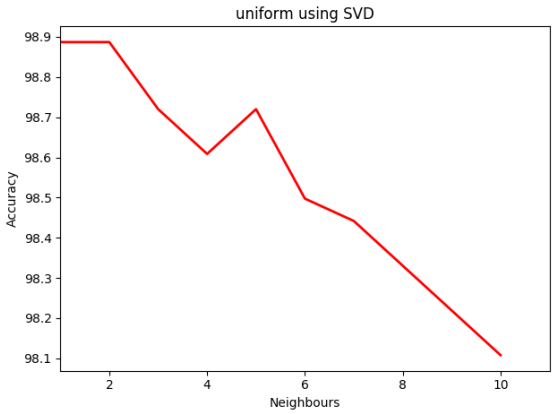
Then have graphics generated using weighted with PCA and SVD methods and uniform distance:

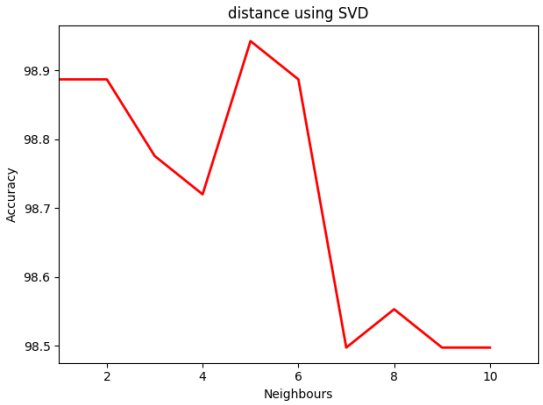
PCA:





SVD:





Marc Llort Maulion (marc.llort)

Conclusions

a) Explanation of the effect of dimensionality KNN

It is important to remember that refer pixel images, so we know that all dimensions refer to pixels.
In this case, to have as many dimensions available can improve accuracy. Also keep in mind however, that from a number of dimensions, not practically improve our accuracy and only the runtime will be higher to finish getting almost the same results, which is why it is important to find the proper relationship.

b) How to understand and understand the results

We have achieved an accuracy of 98% or more when using different algorithms, so I think the results are very good because the number of errors is almost zero and therefore are very accurate.