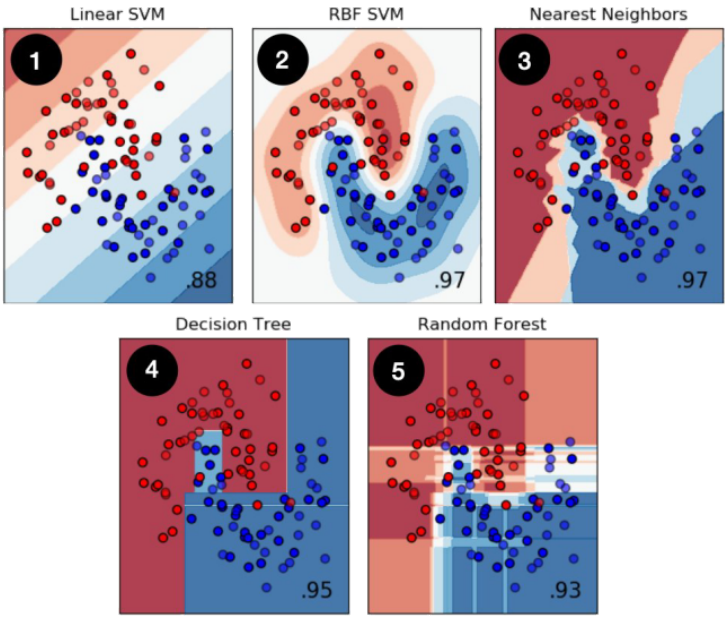
Lab Instructions - session 13

Image Classification

Alireza Kazemipour



Images from scikit-learn documentation, Classifiers Comparison

Note: You will need to have the scikit-learn and the scikit-image libraries installed:
pip install scikit-learn
pip install scikit-image

Classification

A classifier categorizes data into different classes. In this lab we will get familiar with the **Linear and RBF SVM**, **K-Nearest Neighbors,** and **Random Forests** classifiers and apply them on a Persian handwritten digits dataset**.**

Read and Display data

In the folder "**dataset**", you can find 32 by 32 images of handwritten Persian digits. Browse through the subfolders and take a look at the images.

Task 1

Write a program that reads the images from the dataset and randomly displays four examples of each Persian digit.

Linear support vector machine

Suppose your data consist of two classes that are linearly separable, that is the feature vectors of the two classes can be separated by a (hyper)plane. It is possible to categorize new data based on which side of the plane they lie. The SVM method finds this line by maximizing a margin between the two classes.

Run the following code which uses ***raw pixels*** as input features. This means that we simply vectorize each image and feed it directly as the feature vector to the SVM. To handle multiple classes we use ***one vs. one*** approach.

File: **classification.py**

```
import random
from sklearn.preprocessing import StandardScaler
from concurrent import futures
import cv2
from functools import partial
from sklearn.svm import SVC
import numpy as np
import os


def processing(dir, images_list, idx, file):
   print("Processing: " + file)
   temp = []

   for addr in images_list[idx]:
       I = cv2.imread(os.path.join(dir, file, addr))
       temp.append(I.ravel())

   return temp


# Don't bother yourself with this function
def extract_features(dir, images_list, files):
   data = []
   with futures.ProcessPoolExecutor() as executor:
       indices = np.arange(len(images_list))
       func = partial(processing, dir, images_list)
       results = executor.map(func, indices, files)
```

```
        for result in results:
            data.extend(result)
    return data


def main():
    train_dir = './digit_dataset/train/'
    train_labels = []
    train_images_list = []

    files = os.listdir(train_dir)
    files.sort()

    for idx, file in enumerate(files):
        images_list = os.listdir(train_dir + file)
        images_list.sort()
        train_labels.extend([idx] * len(os.listdir(train_dir + file)))
        train_images_list.append(images_list)

    print("------------Feature extraction for train data set------------")
    train_data = extract_features(dir=train_dir, images_list=train_images_list,
files=files)
    print("------------End of extraction-------------------------------")

    # scaler = StandardScaler()
    # train_data = scaler.fit_transform(train_data)

    classifier = SVC(C=0.1, kernel="linear")
    classifier.fit(train_data, train_labels)

    test_dir = './digit_dataset/test/'
    files = os.listdir(test_dir)
    files.sort()

    test_labels = []
    test_images_list = []

    for idx, file in enumerate(files):
        images_list = os.listdir(test_dir + file)
        images_list.sort()
        test_labels.extend([idx] * len(os.listdir(test_dir + file)))
        test_images_list.append(images_list)

    print("------------Feature extraction for test data set------------")
    test_data = extract_features(dir=test_dir, images_list=test_images_list,
files=files)
    print("------------End of extraction-------------------------------")

    print("------------Prediction on train data------------")

    idx = [random.randint(0, len(train_data) - 1) for i in range(10)]
    test_input = [train_data[i] for i in idx]
    test_labels = [train_labels[i] for i in idx]
    # test_input = scaler.fit_transform(test_input)
    results = classifier.predict(test_input)
```

```
    print('predictions: ', results)
    print("train labels: ", list(set(train_labels)))
    print("test labels: ", test_labels)
    print("Accuracy: ", (np.sum(results == test_labels) / len(results)) * 100, "%")


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

- Report the accuracy you obtained?
- Search the scikit learn documentation to find out what a `StandardScaler` object is. Why should it be applied on both training and test data? Uncomment parts of the code where the train and test data are transformed using the Scalar object (through `scaler.fit_transform`) and report the new accuracy.

Task 2

The above accuracy is calculated on the train data. This is not a good way to evaluate a classifier, because it can overfit the training data, that is it can work very well on the training data, but perform poorly on the new (unseen) data. Your task is to examine the accuracy of the classifier both on the **train** and the **test data.**

Radial basis function support vector machine

What if our dataset is not linearly separable? Maybe it is but on a **higher dimensional space**! So we have to map every data point into another high-dimensional space. Here, we will utilize the ***Kernel Trick*** and choose the ***Gaussian RBF*** as the kernel function.

Run the previous code which uses ***raw pixels*** as input features and ***one vs. one*** approach. Replace your classifier and its setting by the following:
(Note that the "*rbf*" is the default kernel of the SVC class.)

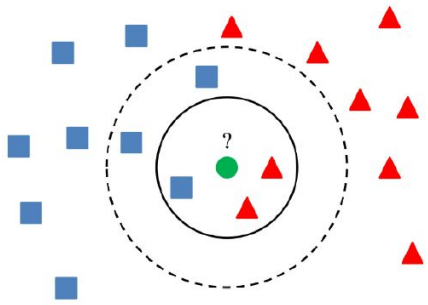
```
classifier = SVC(C=1e+3, gamma=1e-4, kernel="rbf")
```

- What is the regularization parameter `C` and kernel coefficient `gamma`?
- Report your train and test accuracies. Did they improve? Why?

K-Nearest Neighbors

To classify a new data point, one simple solution is to find the K closest feature vectors in the training data and let them vote for the class. In the figure below, if we consider the

3-nearest neighbors, the green circle (new data) is assigned to the *red triangle* category and if we look at the 5-nearest neighbors, it is assigned to the *blue square* category.



https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly%27s_Houses

Run the previous code using the KNN classifier:

```
# Add to your imports
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=1)
```

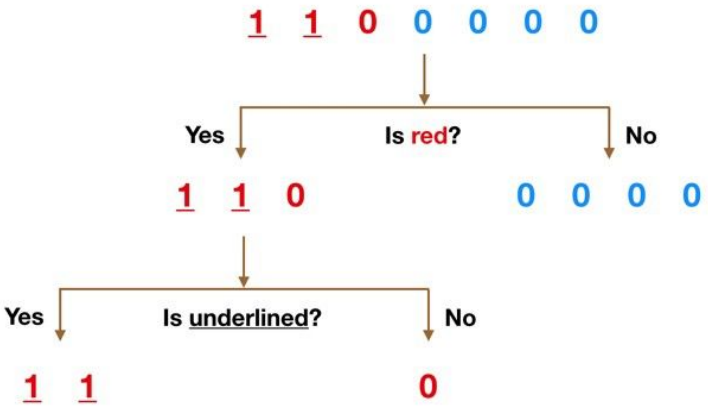
- What do you think is the optimal value of `n_neighbors` for the training accuray? Why?
- What is the optimal value when evaluating on the test data?
- What happens when `n_neighbors` is chosen too large or too small?

Random forests

"Each **random forest** consists of a large number of individual **decision trees** that operate as an *ensemble*; Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction."[1]
So let us first get familiar with decision trees.

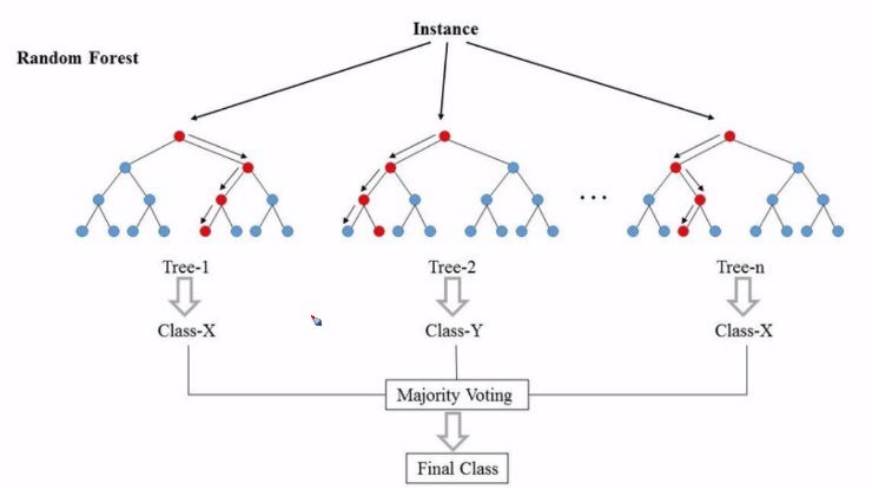
Each decision tree is a series of yes/no questions asked about our data eventually leading to a predicted class. Take a look at the following figure which tries to split its data set into classes that represent each feature.

[1] Understanding random forest



source: Decision trees

So a random forest would look like:



source: https://www.linkedin.com/pulse/random-forest-algorithm-interactive-discussion-niraj-kumar/

Run the previous code using a random forest classifier:

```
# Add to your imports
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=200)
```

- What is the optimal value of `n_estimators`? What happens if we choose a too large or a too small value?

Features

The HOG and LBP features are two common features extracted from an image. They are known to be distinctive, yet robust against image variations. The following code extracts the HOG and LBP features of an image and displays them. Try it on different images and see the results.

File: **display_HOG_featuers.py**

```
from skimage import exposure
from skimage import feature
import cv2
import numpy as np

logo = cv2.imread("01.jpg")
(H, hogImage) = feature.hog(logo, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2),
transform_sqrt=True, block_norm="L1", visualise=True)

hogImage = exposure.rescale_intensity(hogImage, out_range=(0, 255))
hogImage = hogImage.astype("uint8")

cv2.imshow("HOG Image", hogImage)
cv2.waitKey(0)
```

- How do you interpret the displayed HOG features?
- How do you think these lines are related to the image gradients?

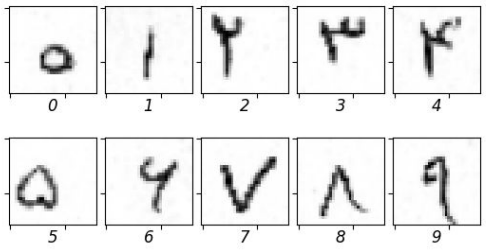
File: **displaying_LBP_features.py**

```
import numpy as np
import cv2
from skimage import feature

fname = '01.jpg'

numPoints = 28
radius = 3

image = cv2.imread(fname, cv2.IMREAD_GRAYSCALE)
lbp = feature.local_binary_pattern(image, numPoints, radius)
J = np.copy(lbp)
J = np.array(J, dtype=np.float32)
cv2.imshow('J', J)
cv2.waitKey(0)
```

- How do you interpret the displayed LBP features?
- Change `numPoints` and `radius`. What happens? Why?

Task 3: classify persian digits using features

You need to classify the given data set in the *digit_dataset* folder using different classifiers and features.



Complete the **processing** and **train** functions in file **task3.py**. Report the **train** and **test accuracies** in a table like below.

file:**task3.py**

```
import random
from sklearn.preprocessing import StandardScaler
from concurrent import futures
import cv2
from functools import partial
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedShuffleSplit
from skimage import feature
import numpy as np
import os

classifier_type = "linear_svm"  # "linear_svm", "rbf_svm", "knn", "random_forest
feature_type = "raw_pixels"  # "raw_pixels", "hog", "lbp", "hog_and_lbp"


# Complete this function!
def processing(feature_type, dir, images_list, idx, file):
   print("Processing: " + file)
   temp = []

   for addr in images_list[idx]:
       I = cv2.imread(os.path.join(dir, file, addr))

       if feature_type == "raw_pixels":
           temp.append(I.ravel())

       elif feature_type == "hog":
           pass
```

```
        elif feature_type == "lbp":
            pass

        elif feature_type == "hog_and_lbp":
            pass

    return temp


# Don't bother yourself with this function.
def extract_features(feature_type, dir, images_list, files):
    data = []
    with futures.ProcessPoolExecutor() as executor:
        indices = np.arange(len(images_list))
        func = partial(processing, feature_type, dir, images_list)
        results = executor.map(func, indices, files)

        for result in results:
            data.extend(result)
    return data


# Complete this function!
def train(classifier_type, train_data, train_labels):
    if classifier_type == "linear_svm":
        pass

    elif classifier_type == "rbf_svm":
        pass

    elif classifier_type == "knn":
        pass

    elif classifier_type == "random_forest":
        pass

    #  Change this line!!!
    return clf.fit(train_data, train_labels)


def main():
    train_dir = './digit_dataset/train/'
    train_labels = []
    train_images_list = []

    files = os.listdir(train_dir)
    files.sort()

    for idx, file in enumerate(files):
        images_list = os.listdir(train_dir + file)
        images_list.sort()
        train_labels.extend([idx] * len(os.listdir(train_dir + file)))
        train_images_list.append(images_list)

    print("------------Feature extraction for train data set------------")
    train_data = extract_features(feature_type=feature_type, dir=train_dir,
```

```
images_list=train_images_list, files=files)
   print("------------End of extraction-------------------------------")

   scaler = StandardScaler()
   train_data = scaler.fit_transform(train_data)

   classifier = train(classifier_type, train_data, train_labels)

   test_dir = './digit_dataset/test/'
   files = os.listdir(test_dir)
   files.sort()

   test_labels = []
   test_images_list = []

   for idx, file in enumerate(files):
       images_list = os.listdir(test_dir + file)
       images_list.sort()
       test_labels.extend([idx] * len(os.listdir(test_dir + file)))
       test_images_list.append(images_list)

   print("------------Feature extraction for test data set------------")
   test_data = extract_features(feature_type=feature_type, dir=test_dir,
images_list=test_images_list, files=files)
   print("------------End of extraction-------------------------------")

   print("------------Prediction on train data-------------")

   idx = [random.randint(0, len(train_data) - 1) for i in range(10)]
   test_input = [train_data[i] for i in idx]
   test_labels = [train_labels[i] for i in idx]
   test_input = scaler.fit_transform(test_input)
   results = classifier.predict(test_input)
   print('predictions: ', results)
   print("train labels: ", list(set(train_labels)))
   print("test labels: ", test_labels)
   print("Accuracy: ", (np.sum(results == test_labels) / len(results)) * 100, "%")

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

Accuracies	Linear SVM	RBF SVM	KNN	Random forest
Raw pixels				
HOG				
LBP				
HOG + LBP				

- What is the most accurate combination? How do you explain this?
- For each classifier or feature type, change the hyperparameters to improve the accuracy. Report what parameters you played with and how they changed the accuracy.

References and further reading

- Local binary patterns
- Different versions of LBP
- HOG-LBP human detector
- Scikit-Learn SVM
- Histogram of oriented gradients
- Grid search
- Random forest
- K-Nearest neighbor
- Udacity SVM course
- Classification by scikit-learn