Lab Four

COMP 219 - Advanced Artificial Intelligence
Cameron Hargreaves, Wei Huang, Gaojie Jin, Xiaowei Huang
University of Liverpool

1 Reading

Begin by reading chapter three of Python Machine Learning until page 80 found in the learning resources section of the vital page for COMP219. Code for this book is available online via the vital page, the book website, or the end of this document, try and go through each line and add comments for what they do

2 Implement the code from the book

1. Implement the perceptron classifier, logistic regression classifier and SVM classifier from the book in your preferred IDE and run these, the full code for this program can be found at the end of this document and online as LabFour-1.py

2.1 Tasks

1. Rename the variable svm to svm_1. Add three more SVM classifiers, svm_2, svm_3, and svm_4 which use the rbf, poly and sigmoid kernels respectively. Modify the code so that we have these in each of our loops and subplots.

2. Re-run the code using the values 0.1, 1, 10 and 100 for gamma on each of your SVMs and see how this affects the decision boundaries and performance of your classifier

3 Creating facial recognition software with SVMs

In this next section we will use the SVM classifier to create a facial recognition software, we will use the olivetti faces dataset, which contains 400 images of 40 people. We will use each pixel value as a learning attribute, and which person it is as our target.

3.1 Implement Initial Code

1. Download the code LabFour-2.py from the course webpage and make sure that it runs. If your version of python is blocked from accessing the internet (via university firewalls,

or any other reason) then download the faces.pkl from the website and replace `faces = fetch_olivetti_faces()` with the below code

```
import pickle
f = open('/PATH/TO/DIRECTORY/DOWNLOADED/FILE/faces.pkl', 'rb')
faces = pickle.load(f)
f.close()
```

2. We can begin exploring the properties of this dataset, by using
`print(faces.images.shape)`
we can see the shape of the image. This image is also stored as a single one dimensional array, where each row is simply concatenated after the other, print the shape of this by showing the shape of faces.data, and see the values of faces.target

3. To get a better understanding of the dataset, print the faces by using our
`print_faces()`
function and passing faces images, target and 400 as our parameters

4. Create an SVC with a linear kernel, and split faces.data and faces.target into X_train, X_test, y_train, y_test using a test_size of 0.25 and a random_state of 0

5. Evaluate the k-fold performance by using the
`evaluate_cross_validation()`
function with our SVC, training and testing data, and five folds as parameters. Read up on this at `http://scikit-learn.org/stable/modules/cross_validation.html` if you are unsure of what's happening. You should get around 95% accuracy

6. Finally test fully by training on the entirety of the training set and testing on the test data using the `train_and_evaluate()` function.

4 Code for SVM classification on Iris Set

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
import numpy as np

import warnings
```

```
warnings.filterwarnings("ignore")

def plot_decision_regions(X, y, classifier,
                test_idx=None, resolution=0.02):

    # setup marker generator and color map

    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                    np.arange(x2_min, x2_max, resolution))

    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot all samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
            alpha=0.8, c=[cmap(idx)],
            marker=markers[idx], label=cl)

    # highlight test samples
    if test_idx:
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1],
            c='', edgecolor='black', alpha=1.0,
            linewidth=1, marker='o',
            s=100, label='test set')

def print_accuracy(name, y_test, y_pred):
    print('Misclassified samples for {0}: {1}'.format(name, (y_test != y_pred).sum()
    print('{0} Accuracy: {1:.2f}'.format(name, accuracy_score(y_test, y_pred)))

iris = datasets.load_iris()
X = iris.data[:, [2,3]]
y = iris.target
```

```
# split into training and test data
X_train, X_test, y_train, y_test = train_test_split(
                                X, y, test_size=0.3, random_state=0)

# define scaler and scale data
sc = StandardScaler()
sc.fit(X_train)

X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

# Define and train perceptron
ppn = Perceptron(n_iter=40, eta0=0.1, random_state=0)
ppn.fit(X_train_std, y_train)

# Define and train logistic regression
lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train_std, y_train)

# Define and train Support Vector Classifier
svm = SVC(kernel='linear', C=1.0, random_state=0)
svm.fit(X_train_std, y_train)

# Create arrays of names and variables for easier looping
classifierName = ['Perceptron', 'Logistic Regression', 'Support Vector Machine']
classifiers = [ppn, lr, svm]
predictions =[]

# make predictions
for clf in classifiers:
    predictions.append(clf.predict(X_test_std))

for i in range(len(classifierName)):
    print_accuracy(classifierName[i], y_test, predictions[i])

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

plt.figure(figsize=(16, 4.8)) # create figure

for i in range(3):
    plt.subplot(1, 3, i + 1) # create a subplot with 3 rows, one column and the i'th
    plot_decision_regions(X_combined_std, y_combined, classifier=classifiers[i], tes
    plt.xlabel('petal length [standardized]')
    plt.ylabel('petal width [standardized]')
```

```
        plt.title(classifierName[i])
        plt.legend(loc='upper left')

    plt.tight_layout()
    plt.show()
```

5 Code for facial recognition software

```
    import sklearn as sk
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces
import pickle
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from scipy.stats import sem
from sklearn import metrics

def print_faces(images, target, top_n):
    # set up the figure size in inches
    fig = plt.figure(figsize=(12, 12))
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1,
                        hspace=0.05, wspace=0.05)
    for i in range(top_n):
        # plot the images in a matrix of 20x20
        p = fig.add_subplot(20, 20, i + 1, xticks=[], yticks=[])
        p.imshow(images[i], cmap=plt.cm.bone)

    # label the image with the target value
    p.text(0, 14, str(target[i]))
    p.text(0, 60, str(i))

def evaluate_cross_validation(clf, X, y, K):
    # create a k-fold cross validation iterator
    cv = KFold(n_splits=K, shuffle=True, random_state=0)
    # by default the score used is the one returned by score method of the estimator (ac
    scores = cross_val_score(clf, X, y, cv=cv)
    print(scores)
    print("Mean score: {0:.3f} (+/-{1:.3f})".format(np.mean(scores), sem(scores)))

def train_and_evaluate(clf, X_train, X_test, y_train, y_test):
    clf.fit(X_train, y_train)

    print("Accuracy on training set:")
```

```
    print(clf.score(X_train, y_train))
    print("Accuracy on testing set:")
    print(clf.score(X_test, y_test))

    y_pred = clf.predict(X_test)

    print("Classification Report:")
    print(metrics.classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(metrics.confusion_matrix(y_test, y_pred))

faces = fetch_olivetti_faces()

# uncomment if firrewall blocks access, download faces.pkl and update the directory path
# f = open('/PATH/TO/DIRECTORY/DOWNLOADED/FILE/faces.pkl', 'rb')
# faces = pickle.load(f)
# f.close()
```