Lab Three

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 68 (p75 2nd edition) 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 and logistic regression 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

2.1 Tasks

1. Modify the code so that instead of inputting the third and fourth columns of the iris dataset (petal length and petal width), use two for loops to test each of the features against each other

2. Using the results from the previous step, which two features give the best performance on training a perceptron classifier, and which two give the best performance for a logistic regression classifier?

3. Update the code to instead use all four features as an input to our classifier (you will have to remove plotting for this)

4. As these are binary classifiers (can only distinguish between two classes), for a multi-class prediction sklearn internally creates three classifiers, and then picks the classifier that has the greatest output. We can see the weights for each of these by printing the coef_ property, each row is the weights for a classifier and each column is the specific weight for a feature. Looking at this for each classifier, which feature is most heavily used to classify the third class, Iris-virginica

```
print(ppn.coef_)

[[-0.08174031  0.06591182 -0.1527238  -0.10880823]
[-0.13903492 -0.20727533  0.36510534 -0.31489372]
[-0.17799526 -0.12336785  0.96486444  0.41486971]]
```

5. In our program we have used a split of 70% of the data to train our classifiers and 30% of the data to test our classifiers. using all four features, loop through the program 100 times with a range of testing data from 1 - 98% of the dataset (hint: look at `np.linspace()`), save the accuracy from each of these runs for the logistic regression classifier, and plot these accuracies. Where do we find peak performance?

3 Further Tasks

1. Use the classification_report function from the sklearn.metrics module using the parameter target_names=iris.target_names to get a more detailed overview of the scores. This uses a confusion matrix approach to the scores which is more widely used for performance metrics, look into what a confusion matrix is.
2. Here we have defined our Scaler and classifiers separately, however this can be time consuming when we are trying out many classifiers, bundle these together using a Pipeline class from sklearn
3. We have used a set random state so that our programs are reproducible across machines. In reality we would want to run the classifier multiple times with different values in our training and test data. Run the logistic regression classifier with five different values of random_state and take the average of their accuracy
4. The above step is a commonly done task called k-folds cross validation, from the sklearn module sklearn.cross_validation import the cross_val_score function and the KFold class, implement a KFold cross validator which shuffles the data 5 times and use cross_val_score to return the average accuracy of these five.

4 Code for sklearn perceptron and logistic regression

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

```
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')

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
sc = StandardScaler()
```

```
sc.fit(X_train)

#scale data
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)

# make predictions
y_pred_ppn = ppn.predict(X_test_std)
y_pred_lr = lr.predict(X_test_std)

print('Misclassified samples for perceptron: {0}'
    .format((y_test != y_pred_ppn).sum()))
print('Perceptron Accuracy: {0:.2f}'
    .format(accuracy_score(y_test, y_pred_ppn)))

print('Misclassified samples for Logistic Regression: {0}'
    .format((y_test != y_pred_lr).sum()))
print('Logistic Regression Accuracy: {0:.2f}'
    .format(accuracy_score(y_test, y_pred_lr)))

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

# plot_decision_regions(X_combined_std,
#                        y_combined,
#                        classifier=ppn,
#                        test_idx=range(105,150))

plot_decision_regions(X_combined_std,
                      y_combined,
                      classifier=lr,
                      test_idx=range(105, 150))

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```