Lab Five

COMP 219 - Advanced Artificial Intelligence

Xiaowei Huang
Cameron Hargreaves

October 29th 2018

1 Decision Trees and Random Forests

1.1 Reading

Begin by reading chapter three of Python Machine Learning page 80 to 92 found in the learning resources section of the vital page for COMP219

1.2 Implement the Decision Tree and Random Forest Classifier

1. For the next piece of code we will be using a package that does not come bundled with Anaconda, GraphViz. Due to user privileges you are unable to install this directly on to the university machines, however we can install this into a virtual environment. Begin by opening the Anaconda prompt found in the start menu (if you are using your own computer and have Anaconda installed the below command should just work from the terminal/command line)
   `conda create --name comp219 python=3.7 scikit-learn python-graphviz matplotlib`
   `activate comp219` (or on mac/linux) `source activate comp219`

2. Download the code from the course website and navigate to the folder this is in (in the Anaconda prompt), run it using `python LabFive-1.py` you will be able to see how the decision regions for decision trees and random forest differ from the classifiers we have previously implemented, additionally we will have created a file called iris.pdf. If we open this then we can see how the decision tree has made its' classification boundaries.

1.3 The Titanic dataset

For this next task we will use the decision tree classifier to predict which passengers of the Titanic survived, based on their age, passenger class and sex.

1.4 Tasks

1. Activate the virtual environment we created in the previous step

2. From here open the file LabFive-2.py, change the path to the location of the downloaded titanic.csv file and ensure that we can load this as we would expect by running the code, we can verify this by using
   `print(feature_names)`
   `print(titanic_X[0], titanic_y[0])`

3. First select the attributes that we will use for learning, use array indexing to select the second, fifth and eleventh columns of our feature_names and titanic_X arrays (indexes [1, 4, 11]), corresponding to class, age and sex.

4. Print the 13th row (index 12) of titanic_X what could be problematic with this record

5. Remedy the above problem by calculating the average value of this column and replacing all 'NA' with the average age

6. Two of our features, class and sex, are text based categorical features, where we wish them to be integer values. Using the LabelEncoder from sklearn.preprocessing, fit each of these in turn, transform them and reassign to the column so that we just have integer values

7. Now that we have pre-processed the data so that it is in a usable form we can apply our decision tree to this dataset, use train_test_split() from sklearn.cross_validation, using a test size of 0.25 and a random state of 33, to split our data into training and testing data.

8. Create a new decision tree classifier with a max_depth of 3, min_samples_leaf of 5 and criterion of 'entropy', and fit our training data to this.

9. Use graphviz to generate a pdf of our decision tree and evaluate. At each node we can see how many died (0) or survived (1) by looking in the Value field. Then we traverse the tree by answering the question that we have in the first line in each node and going left for true and right for false. For example if we were to look at females, then we know the sex is represented by integer 0 and is therefore less than 0.5. Of our original 984 samples, 333 of these were female.

10. Which group of people from this tree had the highest probability of survival, and which had the lowest.

2 The K-Means Clustering Algorithm

2.1 Reading

Begin by reading chapter three of Python Machine Learning page 80 to 92 found in the learning resources section of the vital page for COMP219 from page 92 until the end of the chapter.

2.2 Implement the K-Means Clustering Algorithm

1. Run the code LabFive-3.py from the course website and run to see how the decision regions vary for the K-Means clustering algorithm. Alter the input features and see how the decision regions vary for different numbers of clusters.

2. The K-Means clustering algorithm is very simple to implement yourself, take the code from LabFive-4.py and run it. Next un-comment lines 8-28 and fill in code for the values of labels, new_centers and centers, as well as the condition for the if statement with your own implementation. Then un-comment the final three lines. The two plots returned by the program should be identical.

3 Further Work

1. For the titanic dataset we used the LabelEncoder, this is generally fine for binary values, however for categorical fields with more than two values this can lead to issues where we can assign numerical significance to features where there is none. Reimplement this by adding three more fields and using One Hot encoding instead.

2. Pick another dataset and apply the K-Means clustering algorithm to this.
https://archive.ics.uci.edu/ml/index.php is an excellent resource for finding datasets to test out new algorithms.

4 Code for Decision Tree/Random Forest 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.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz
import graphviz

import io

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)

# Define and train our decision tree classifier
tree = DecisionTreeClassifier(criterion='entropy',
                              max_depth=3,
                              random_state=0)
tree.fit(X_train, y_train)

# Define and train our random forest classifier
forest = RandomForestClassifier(criterion='entropy',
                                n_estimators=10,
                                random_state=1,
                                n_jobs=2)
forest.fit(X_train, 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))

predictions.append(tree.predict(X_test))

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

print_accuracy('Random Forest', y_test, predictions[3])

X_combined = np.vstack((X_train, X_test))
# Scaling is not a requirement for decision trees
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(len(classifierName)):
    plt.subplot(1, 5, i + 1) # create a subplot with 4 rows,
                             # one column and the i'th member
    plot_decision_regions(X_combined_std, y_combined,
        classifier=classifiers[i], test_idx=range(105, 150))
    plt.xlabel('petal length [standardized]')
    plt.ylabel('petal width [standardized]')
    plt.title(classifierName[i])
    plt.legend(loc='upper left')

# create a subplot on the fourth member for our Random Forest classifier
plt.subplot(1, 5, 4)
plot_decision_regions(X_combined, y_combined, classifier=tree,
    test_idx=range(105, 150))
plt.title('Decision Tree')

# create a subplot on the fourth member for our Random Forest classifier
plt.subplot(1, 5, 5)
plot_decision_regions(X_combined, y_combined, classifier=forest,
    test_idx=range(105, 150))
plt.title('Random Forest')

plt.tight_layout()
plt.show()

dot_data = export_graphviz(tree, out_file=None,
    feature_names=iris.feature_names[2:4],
```

```
        class_names=iris.target_names)
graph = graphviz.Source(dot_data)
graph.render('iris')
```

5 Initial Code for Titanic Dataset with Decision Trees

```
import csv
import numpy as np

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz

import graphviz

pathToTitanic = '/YOUR/PATH/TO/titanic.csv'

with open(pathToTitanic, 'rt') as csvfile:
    titanic_reader = csv.reader(csvfile, delimiter=',',
        quotechar='"', quoting=csv.QUOTE_ALL)

    # Header contains feature names
    row = next(titanic_reader)
    feature_names = np.array(row)

    # Load dataset, and target classes
    titanic_X, titanic_y = [], []
    for row in titanic_reader:
        titanic_X.append(row)
        titanic_y.append(row[2]) # The target value is "survived"

    titanic_X = np.array(titanic_X)
    titanic_y = np.array(titanic_y)

print(titanic_X[0], titanic_y[0])
```

6 Code for K-Nearest Neighbours

```
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.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz
from sklearn.neighbors import KNeighborsClassifier

import graphviz

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)

# Define and train our decision tree classifier
tree = DecisionTreeClassifier(criterion='entropy',
                                max_depth=3,
                                random_state=0)
tree.fit(X_train, y_train)
```

```
# Define and train our random forest classifier
forest = RandomForestClassifier(criterion='entropy',
                                n_estimators=10,
                                random_state=1,
                                n_jobs=2)
forest.fit(X_train, y_train)

knn = KNeighborsClassifier(n_neighbors=20, p=2, metric='minkowski')
knn.fit(X_train_std, y_train)

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

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

predictions.append(tree.predict(X_test))

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

print_accuracy('Random Forest', y_test, predictions[3])

X_combined = np.vstack((X_train, X_test))
# Scaling is not a requirement for decision trees
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(len(classifierName)):
    plt.subplot(1, 6, i + 1)
    # create a subplot with 4 rows, one column and the i'th member
    plot_decision_regions(X_combined_std, y_combined,
        classifier=classifiers[i], test_idx=range(105, 150))
    plt.xlabel('petal length [standardized]')
    plt.ylabel('petal width [standardized]')
    plt.title(classifierName[i])
    plt.legend(loc='upper left')
```

```
# create a subplot on the fourth member for our Random Forest classifier
plt.subplot(1, 6, 5)
plot_decision_regions(X_combined, y_combined, classifier=tree,
    test_idx=range(105, 150))
plt.title('Decision Tree')

# create a subplot on the fourth member for our Random Forest classifier
plt.subplot(1, 6, 6)
plot_decision_regions(X_combined, y_combined, classifier=forest,
    test_idx=range(105, 150))
plt.title('Random Forest')

plt.tight_layout()
plt.show()

dot_data = export_graphviz(tree, out_file=None,
    feature_names=iris.feature_names[2:4],
    class_names=iris.target_names)
graph = graphviz.Source(dot_data)
graph.render('iris')
```