Understanding the Art of Voting: A Visual Approach

Experimentation & Verification on the Performance of a Voting Classifier

Motivation

Ensemble method can also be known as the wisdom of the crowd, in which a conclusion is made based on answers given by multiple parties. A classifier for an ensemble method can be made from multiple different machine learning algorithms of the user's liking. One such classifier, known as soft voting classifier, first averages out the prediction probability from each individual classifier before predicting the class based on the highest probability. It is believed using a voting classifier could outperform the classifiers constituting it. This is due to the law of large numbers, in which having many bias classifiers predicting a class will converge the prediction to that class, in other words, the probability of predicting a majority class is higher. The downside is if the majority predicted class is wrong, error is propagated to the voting classifier. If each classifier has diverse performance and error, the same error will not compound in the voting classifier and shouldn't deteriorate the voting classifier's performance. Hence, in this project, experiment is done to evaluate the performance of a soft voting classifier.

Methodology

The machine learning task to work on would be a binary classification task. Three completely different machine learning algorithms for diversity, namely logistic regression (LR), decision trees (DT) and multi-layer perceptron (MLP) will be trained separately and compared with the soft voting classifier (VC), which is made up of those three machine learning algorithms. The dataset to be used will be the chronic kidney disease dataset from the UCI Machine Learning Repository. The experiment is outlined as follows. First, pre-process the dataset by only using numerical attributes. The dataset is then split randomly for 60% training data and 40% testing data. For the visualization of the classification boundary, only two attributes will be used and they are the top 2 most important attributes shared commonly among each algorithm using Recursive Feature Elimination. Each classifier is fitted with the training dataset and the training accuracy is obtained which is compared with its test accuracy in a bar chart. A contour plot describing the classification regions for each classifier and where those test data lie at using the two attributes from the dataset is plotted for analysis.

Results

Figure 1 shows the bar chart comparing the training and testing accuracy for each classifier.

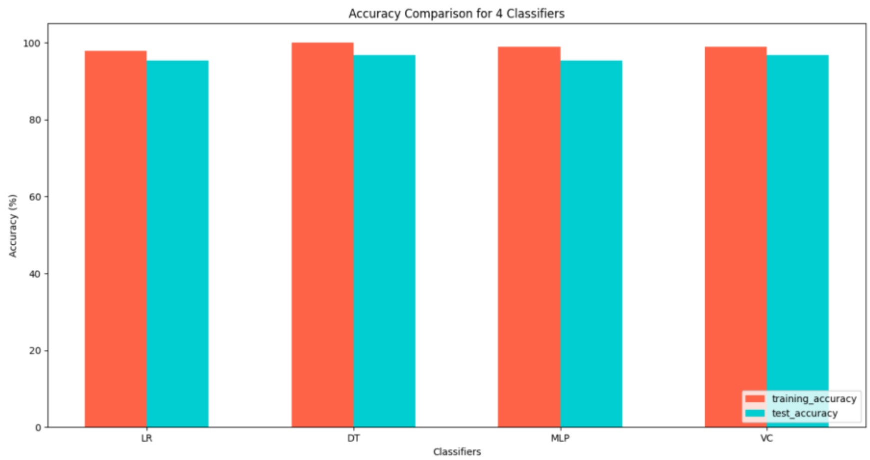

Figure 1: Accuracy Comparison for the Four Classifiers.

Figure 2 shows the classification region or the decision boundary of each classifier. The data points here are from the test set (40% of the whole dataset).

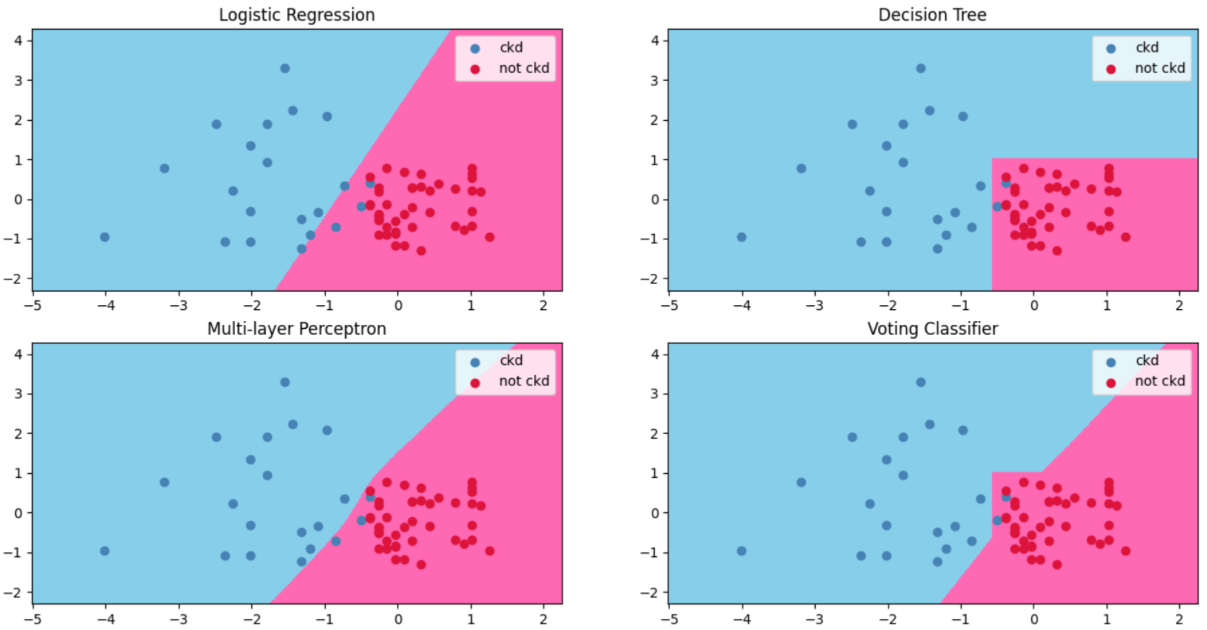

Figure 2: Classification region for all four classifiers. Chronic kidney disease (ckd) is the blue region whereas the opposite class (not ckd) is the pink region.

Table 1 explicitly states the training and testing accuracy values and their differences for each classifier.

Table 1: Comparison of training and test accuracies with their differences for each classifier

Classifier	Training Accuracy (%)	Test Accuracy (%)	Difference (%)
Logistic Regression	97.872	95.312	2.560
Decision Tree	100.00	96.875	3.125
Multi-layer Perceptron	98.936	95.313	3.623
Soft Voting Classifier	98.936	96.875	2.061

Based on Figure 2, it can be seen that the three non-fusion classifiers all have their own decision boundary, in which LR has a linear classification region whereas DT and MLP have a non-linear classification region. Due to the linearity of the LR, some of the ckd samples are predicted incorrectly. Despite that, MLP, which is non-linear, still incorrectly predicted some of the test samples, making it performed similarly to LR. On the other hand, the box-shaped boundary region of DT allowed DT to prevent misclassifying the test samples like LR and MLP. As for VC, it is noted that VC contains the characteristics of DT and MLP. However, the stretched-out classification region of the VC differed in a way that this region is slightly shifted to the right, making the VC not misclassifying the test sample like that in MLP. Based on each classifier's classification region, no traces of over-/under-fitting can be observed. It can be inferred that the performance of VC is highly dependent on the classifiers constituting it. If those classifiers are similar in performance and classification region, the VC might not perform exceptionally well. For example, during training, DT scored perfectly but not for LR and MLP, and this error was propagated to the VC, as seen in Table 1 whereby the VC's training accuracy isn't perfect, but better than LR. Therefore, VC might be able to perform remarkably well than the individual classifiers constituting it if there are sufficient number of those classifiers with good performances and diverse decision boundaries.

Appendix : Source Code

37215037_LokTzeLun_FinalProject.py

```python
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import VotingClassifier
import numpy as np
import pandas as pd


# Loads the dataset given the path and the filename
def load(filepath):
    data = []
    f = open(filepath, "r")
    lines = f.read().split("\n")  # split the dataset according to the lines/rows
    for line in lines:
        data.append(line.split(","))  # split a line at each comma
    f.close()
    return data


# Filter out non-numerical attributes from the dataset and process the class labels
def filter_numerical_attributes_and_class_label(data):
    numerical_col_ind = [0, 1, 9, 10, 11, 12, 13, 14, 15, 16, 17]  # all the 11 numeric attributes column index
    df = pd.DataFrame(data)
    class_label = df.iloc[:, 24]
    class_label = class_label.replace("ckd", 0)
    class_label = class_label.replace("notckd", 1)
    df = df.iloc[:, numerical_col_ind]
    return df.to_numpy(dtype=float), class_label.to_numpy()


# For analysis purpose, get the index of rows with missing data. Missing data is represented as '?' in the columns
def get_index_of_missing_data(data):
    ind_list = []
    for idx, d in enumerate(data):
        if '?' in d:
            ind_list.append(idx)
    return ind_list


# remove rows with missing data
def remove_rows_of_missing_data(data):
    filtered = []
    for d in data:
        if '?' not in d:
            filtered.append(d)
    return filtered


```

```python
# simple function to calculate the accuracy using classifiers and datasets
def get_classifier_accuracy(classifier, x, y):
    # get predictions
    y_predict = classifier.predict(x)
    accuracy = 0.0
    for index, actual in enumerate(y):
        if y_predict[index] == actual:
            accuracy += 1.0
    return accuracy / float(len(y)) * 100.0


# function to plot the classification region for any classifier
def plot_clf_region(classifier, x, y):
    # first get the bounds of the domain for the contour plot, this would be the max and min from the two attributes
    min1, max1 = x[:, 0].min() - 1, x[:, 0].max() + 1
    min2, max2 = x[:, 1].min() - 1, x[:, 1].max() + 1

    # create ranges of data for attributes x1 and x2
    res = 0.01  # resolution of the contour plot
    x1 = np.arange(min1, max1, res)
    x2 = np.arange(min2, max2, res)

    # forming a grid, basically means one row of x1 for each point on the y-axis and one column of x2 for each point
    # on the x-axis
    xx, yy = np.meshgrid(x1, x2)

    # flatten the grid to allow the samples from it to be used for prediction by the classifier
    r1, r2 = xx.flatten(), yy.flatten()
    r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))

    # stack the r1 and r2 vectors horizontally to create [attribute1, attribute2] format for classifier use
    grid = np.hstack((r1, r2))

    # use the classifier to predict all the points in the grid
    y_predict = classifier.predict(grid)

    # reshape the prediction vector back to mesh-grid format for plotting purposes
    z = y_predict.reshape(xx.shape)

    # finally plot it using matplotlib's filled contour plot
    cmap_contour = ListedColormap(['skyblue', 'hotpink'])
    plt.contourf(xx, yy, z, cmap=cmap_contour)

    # also add the test data points to it
    label = ['ckd', 'not ckd']
    c_scatter = ['steelblue', 'crimson']
    for class_label in range(2):
        # get the rows of the test data that is of class_label 0 or 1
        row = np.where(y == class_label)
        plt.scatter(x[row, 0], x[row, 1], label=label[class_label], c=c_scatter[class_label])
    plt.legend(loc="best")


# dataset used is the chronic kidney disease classification dataset from UCI Machine Learning Repository
# this dataset has 11 numerical attributes and 13 nominal attributes (24 attributes in total, excluding class label)
# path that leads to the location of the dataset (stored locally in own device)
pathname = "Chronic_Kidney_Disease/chronic_kidney_disease_data.txt"
```

37215037_LokTzeLun_FinalProject.py

```python
    # load the dataset into a list
    dataset = load(pathname)
    print("Visualize the first 5 data")
    print(dataset[0:5])
    print("\nNumber of data : ", len(dataset))  # there would be 400 number of data in total
    # get the indices of rows with missing data, a row has missing data if it has a '?' in at least one column
    row_miss = get_index_of_missing_data(dataset)
    print("\nNumber of data with missing columns : ", len(row_miss))  # there would be 242 number of unusable data
    dataset = remove_rows_of_missing_data(dataset)
    print("\nNumber of data after filtering out missing columns : ", len(dataset))  # only 158 number of useful data

    # since in this project, only numerical attributes are used. Filter out non-numerical attributes
    # also separate and process the class labels, ckd and notckd as 0 and 1 respectively
    X, Y = filter_numerical_attributes_and_class_label(dataset)
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.4, random_state=40)

    # Scale the data attributes to their unit variance
    scale = StandardScaler()
    x_train = scale.fit_transform(x_train)
    x_test = scale.transform(x_test)

    # Initialize each classifier: Logistic regression, decision tree, multi-layer perceptron and voting classifier
    lr = LogisticRegression(random_state=10)
    dt = DecisionTreeClassifier(random_state=12)
    mlp = MLPClassifier(random_state=30)
    vc = VotingClassifier(estimators=[('lr', lr), ('dt', dt), ('mlp', mlp)], voting='soft')

    # Perform feature selection on the dataset for each classifier using Recursive Feature Selector
    # Unfortunately, MLPClassifier isn't supported by RFE function. Instead of changing the classifier used,
    # only the logistic regression and the decision tree classifier is used to obtain the 2 important attributes
    n_features = 4
    selector_lr = RFE(estimator=lr, n_features_to_select=n_features, step=1)  # for logistic regression
    selector_lr = selector_lr.fit(x_train, y_train)
    selector_dt = RFE(estimator=dt, n_features_to_select=n_features, step=1)  # for decision tree classifier
    selector_dt = selector_dt.fit(x_train, y_train)

    # Visualize and look for the 2 common important features for all classifiers
    print("Ranking of Features for LR : ", selector_lr.ranking_)
    print("Ranking of Features for DT : ", selector_dt.ranking_)

    # as the classifiers differ immensely, selecting the two attributes is based on own judgement
    # from own perspective, it seems col 8 and col 9 will be the 2 attributes chosen to proceed with the experiment
    # changing the random state of each classifier and the train_test_split func might alter this decision!
    feature_to_keep_idx = [8, 9]

    # Reduce the dataset to only two of the selected attributes
    new_x_train = x_train[:, feature_to_keep_idx]  # x_train now only contains the two selected feature
    new_x_test = x_test[:, feature_to_keep_idx]  # x_test now only contains the two selected feature

    # Train each classifier
    lr.fit(new_x_train, y_train)
    dt.fit(new_x_train, y_train)
    mlp.fit(new_x_train, y_train)
    vc.fit(new_x_train, y_train)

    # Retrieve their training accuracy
    train_acc_lr = get_classifier_accuracy(lr, new_x_train, y_train)
    train_acc_dt = get_classifier_accuracy(dt, new_x_train, y_train)
    train_acc_mlp = get_classifier_accuracy(mlp, new_x_train, y_train)
    train_acc_vc = get_classifier_accuracy(vc, new_x_train, y_train)
    print("Training Accuracy for Logistic Regression : ", train_acc_lr)
```

```python
print("Training Accuracy for Decision Tree : ", train_acc_dt)
print("Training Accuracy for Multilayer Perceptron : ", train_acc_mlp)
print("Training Accuracy for Voting Classifier : ", train_acc_vc)

# Use the classifiers to make predictions on test set and get the test accuracy
test_acc_lr = get_classifier_accuracy(lr, new_x_test, y_test)
test_acc_dt = get_classifier_accuracy(dt, new_x_test, y_test)
test_acc_mlp = get_classifier_accuracy(mlp, new_x_test, y_test)
test_acc_vc = get_classifier_accuracy(vc, new_x_test, y_test)
print("\nTest Accuracy for Logistic Regression : ", test_acc_lr)
print("Test Accuracy for Decision Tree : ", test_acc_dt)
print("Test Accuracy for Multilayer Perceptron : ", test_acc_mlp)
print("Test Accuracy for Voting Classifier : ", test_acc_vc)

# Comparison of each classifier with their training and test accuracy using a bar chart
train_acc = np.array([train_acc_lr, train_acc_dt, train_acc_mlp, train_acc_vc])  # height of training accuracy
test_acc = np.array([test_acc_lr, test_acc_dt, test_acc_mlp, test_acc_vc])  # height of test accuracy
bar_width = 0.3  # the width of each bar in the bar chart
number_of_bars = 4  # since we have 4 classifiers
ind = np.arange(number_of_bars)  # index position for the bar chart

plt.bar(ind, train_acc, color='tomato', width=bar_width, label='training_accuracy')
plt.bar(ind + bar_width, test_acc, color='darkturquoise', width=bar_width, label='test_accuracy')
plt.xlabel('Classifiers')
plt.ylabel('Accuracy (%)')
plt.title('Accuracy Comparison for 4 Classifiers')
plt.xticks(ind + bar_width / 2, ('LR', 'DT', 'MLP', 'VC'))
plt.legend(loc='lower right')
plt.show()

# Visualization of each classifier's classification boundary using a contour plot
plt.subplot(2, 2, 1)
plot_clf_region(lr, new_x_test, y_test)
plt.title("Logistic Regression")
plt.subplot(2, 2, 2)
plot_clf_region(dt, new_x_test, y_test)
plt.title("Decision Tree")
plt.subplot(2, 2, 3)
plot_clf_region(mlp, new_x_test, y_test)
plt.title("Multi-layer Perceptron")
plt.subplot(2, 2, 4)
plot_clf_region(vc, new_x_test, y_test)
plt.title("Voting Classifier")
plt.show()
```