projeto 4

July 29, 2024

Stochastic Processes - class 1/2024 - University of Brasília Computational work 4 - classification

Gabriel Tambara Rabelo - 241106461

O presente projeto visa classificar de diferentes métodos para

expectancy,mode,median,variance,skewness,kurtosis,entropy,class 53.891967984934084,16.0,16.0,2366.49105036432
80.08158192090394,16.0,48.0,5663.520651358167,289773.10140834015,61596894.36590457,2.3554137854502386,0
80.37126177024481,16.0,48.0,6078.010526279875,372971.9596642113,79378789.52637905,2.392316701226323,0

h[0],h[1],h[2],h[3],h[4],h[5],h[6],h[7],class 58419,11150,9625,13219,13300,450,20,17,0
52196,7232,6733,7406,9320,9624,12177,1512,0

Para uma rodada dos testes do código, foram geradas as matrizes de confusão e suas análises indicam alguns pontos de interesse quanto ao método e o conjunto específico de dados utilizado. Esses resultados obtidos não necessariamente indicam que um algoritmo é melhor que o outro, contudo, para o presente conjunto de dados e amostras, podemos definir sua classificabilidade.

A seguir, têm-se os resultados de um dos testes da classificação, seguido pela sua análise.

——————CASE1——————

Bayes Confusion Matrix: [[11 0 0 0 0] [0 1 0 1 0] [0 0 3 0 0] [0 0 0 6 0] [0 0 0 0 4]] Bayes Accuracy per Class: [1. 0.5 1. 1. 1.] Bayes Accuracy: 0.9615384615384616

QDA Confusion Matrix: [[11 0 0 0 0] [0 2 0 0 0] [0 0 3 0 0] [0 0 0 6 0] [0 0 0 0 4]] QDA Accuracy per Class: [1. 1. 1. 1. 1.] QDA Accuracy: 1.0

LDA Confusion Matrix: [[11 0 0 0 0] [0 0 0 0 2] [0 0 3 0 0] [0 0 0 6 0] [0 0 2 0 2]] LDA Accuracy per Class: [1. 0. 1. 1. 0.5] LDA Accuracy: 0.8461538461538461

——————CASE2——————

Bayes Confusion Matrix: [[11 0 0 0 0] [0 1 0 1 0] [0 0 3 0 0] [0 1 0 4 1] [0 0 0 0 4]] Bayes Accuracy per Class: [1. 0.5 1. 0.66666667 1.] Bayes Accuracy: 0.8846153846153846

QDA Confusion Matrix: [[11 0 0 0 0] [0 1 0 1 0] [0 0 3 0 0] [0 0 0 4 2] [0 0 0 0 4]] QDA Accuracy per Class: [1. 0.5 1. 0.66666667 1.] QDA Accuracy: 0.8846153846153846

LDA Confusion Matrix: [[11 0 0 0 0] [0 0 1 1 0] [0 0 3 0 0] [0 0 0 6 0] [0 0 1 2 1]] LDA Accuracy per Class: [1. 0. 1. 1. 0.25] LDA Accuracy: 0.8076923076923077

Para o caso 1:

O classificador Naive Bayes teve um bom desempenho para a maioria das classes, com uma precisão perfeita para 4 das 5 classes. A classe 2 teve um desempenho menor com precisão de 50%, indicando que há espaço para melhorias na diferenciação dessa classe. A precisão geral é alta, mostrando que o classificador funciona bem com esses dados, mas pode ter limitações em situações onde as classes não são bem separadas. Esse problema seria resolvido com um espaço amostral maior, o que poderia gerar melhores acurácias também para a classe problemática, porém, também poderia reduzir a acurácia da classificação das outras classes.

O QDA obteve uma precisão perfeita em todas as classes. Isso indica que o QDA foi capaz de capturar muito bem as características dos dados, provavelmente devido à sua capacidade de modelar distribuições não lineares. No entanto, precisão perfeita pode às vezes indicar overfitting, especialmente com um conjunto de dados de teste pequeno.

O LDA teve uma boa performance para a maioria das classes, mas falhou completamente para a classe 2, e teve uma precisão de 50% para a classe 4. A precisão geral é inferior ao Naive Bayes e QDA.

Para o caso 2:

O Naive Bayes apresentou uma redução na precisão geral e por classe comparado ao caso 1. A classe 2 ainda tem uma precisão de 50%, e a classe 4 reduziu para 66.67%.

O QDA também apresentou uma redução na precisão geral e por classe comparado ao caso 1. A precisão para a classe 4 diminuiu para 66.67%, e a classe 2 manteve a precisão de 50%..

O LDA teve uma redução significativa na precisão geral e para a classe 4 comparado ao Case 1. A precisão para a classe 2 é nula e a precisão para a classe 4 é de apenas 25%, indicando dificuldades significativas em separar essas classes com as características usadas. A precisão geral é a mais baixa entre os classificadores, indicando que o LDA é o menos eficaz com as características do caso 2.

A seguir, segue o código equivalente.

[ ]:
```
import os
import csv
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.preprocessing import StandardScaler

img_size = 300
classes = ['Alzheimer', 'COVID', 'Brazilian_seeds', 'Brazilian_leaves',␣
↪'skin_cancer']
data_dir = './image_dataset/'
case1_csv_filename = 'case1_statistics.csv'
case2_csv_filename = 'case2_statistics.csv'
reg_param = 1e-5  # Adjusted regularization parameter
```

```
def load_images_from_folder(folder_path):
    image_files = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.
↪join(folder_path, f))]
    images = []

    for image_file in image_files:
        img = cv.imread(os.path.join(folder_path, image_file), cv.
↪IMREAD_GRAYSCALE)
        ratio = img_size / img.shape[1]
        img_resized = cv.resize(img, (img_size, int(img.shape[0] * ratio)), cv.
↪INTER_AREA)
        images.append(img_resized)

    return images

def makeHist(image, bits):
    hist, bins = np.histogram(image.flatten(), bins=np.arange(0, 257, 2 ** (8 -␣
↪bits)))
    return hist, bins

def normalizeHist(hist):
    total_pixels = np.sum(hist)
    normalized_hist = hist / total_pixels
    return normalized_hist

def bin_centers(bin_borders):
    bin_center_list = (bin_borders[:-1] + bin_borders[1:]) / 2
    bin_center_list[-1] = 255
    return bin_center_list

def expectancy(hist, bin_centers):
    return np.sum(hist * bin_centers)

def median(hist, bin_centers):
    cumulative_freq = 0
    for i, freq in enumerate(hist):
        cumulative_freq += freq
        if cumulative_freq >= 0.5:
            return bin_centers[i]

def mode(hist, bin_centers):
    return bin_centers[np.argmax(hist)]

def moment(hist, bin_centers, order, expectancy):
    return np.sum(((bin_centers - expectancy) ** order) * hist)
```

```python
def entropy(hist):
    epsilon = 1e-27
    return -np.sum(hist * np.log2(hist + epsilon))

def calculate_statistics(image, bins):
    hist, bin_edges = makeHist(image, bins)
    normalized_hist = normalizeHist(hist)
    centered_bins = bin_centers(bin_edges)

    var_exp = expectancy(normalized_hist, centered_bins)
    var_median = median(normalized_hist, centered_bins)
    var_mode = mode(normalized_hist, centered_bins)
    var_variance = moment(normalized_hist, centered_bins, 2, var_exp)
    var_skewness = moment(normalized_hist, centered_bins, 3, var_exp)
    var_kurtosis = moment(normalized_hist, centered_bins, 4, var_exp)
    var_entropy = entropy(normalized_hist)

    return hist, var_exp, var_mode, var_median, var_variance, var_skewness,
 ↪var_kurtosis, var_entropy

def save_statistics_to_csv(images, class_label, case1_csv_filename,
 ↪case2_csv_filename):
    with open(case1_csv_filename, mode='a', newline='') as case1_file,
 ↪open(case2_csv_filename, mode='a', newline='') as case2_file:
        case1_writer = csv.writer(case1_file)
        case2_writer = csv.writer(case2_file)

        for image in images:
            hist, var_exp, var_mode, var_median, var_variance, var_skewness,
 ↪var_kurtosis, var_entropy = calculate_statistics(image, bins=8)

            case_1_data = list(hist) + [class_label]
            case_2_data = [var_exp, var_mode, var_median, var_variance,
 ↪var_skewness, var_kurtosis, var_entropy, class_label]

            case1_writer.writerow(case_1_data)
            case2_writer.writerow(case_2_data)

def load_data_from_csv(filename):
    data = []
    labels = []
    with open(filename, mode='r') as file:
        reader = csv.reader(file)
        next(reader)
        for row in reader:
            data.append([float(x) for x in row[:-1]])
            labels.append(int(row[-1]))
```

```
    return np.array(data), np.array(labels)

def train_test_split(data, labels, test_size=0.1):
    np.random.seed(np.random.randint(0,99))
    indices = np.arange(len(data))
    np.random.shuffle(indices)
    split_idx = int(len(data) * (1 - test_size))
    train_indices = indices[:split_idx]
    test_indices = indices[split_idx:]
    return data[train_indices], data[test_indices], labels[train_indices],␣
↪labels[test_indices]

with open(case1_csv_filename, mode='w', newline='') as case1_file,␣
↪open(case2_csv_filename, mode='w', newline='') as case2_file:
    case1_writer = csv.writer(case1_file)
    case2_writer = csv.writer(case2_file)

    case1_writer.writerow([f'h[{i}]' for i in range(8)] + ['class'])
    case2_writer.writerow(['expectancy', 'mode', 'median', 'variance',␣
↪'skewness', 'kurtosis', 'entropy', 'class'])

for class_id, class_name in enumerate(classes):
    folder_path = os.path.join(data_dir, class_name)
    images = load_images_from_folder(folder_path)
    save_statistics_to_csv(images, class_id, case1_csv_filename,␣
↪case2_csv_filename)

data_case1, labels_case1 = load_data_from_csv(case1_csv_filename)
data_case2, labels_case2 = load_data_from_csv(case2_csv_filename)

X_train_case1, X_test_case1, y_train_case1, y_test_case1 =␣
↪train_test_split(data_case1, labels_case1, test_size=0.1)
X_train_case2, X_test_case2, y_train_case2, y_test_case2 =␣
↪train_test_split(data_case2, labels_case2, test_size=0.1)

class NaiveBayesClassifier:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = {}
        self.var = {}
        self.priors = {}

        for cls in self.classes:
            X_c = X[y == cls]
            self.mean[cls] = np.mean(X_c, axis=0)
            self.var[cls] = np.var(X_c, axis=0) + reg_param
```

```
            self.priors[cls] = X_c.shape[0] / X.shape[0]

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        posteriors = []

        for cls in self.classes:
            prior = np.log(self.priors[cls])
            posterior = np.sum(np.log(self._pdf(cls, x)))
            posterior = prior + posterior
            posteriors.append(posterior)

        return self.classes[np.argmax(posteriors)]

    def _pdf(self, cls, x):
        mean = self.mean[cls]
        var = self.var[cls]
        numerator = np.exp(- (x - mean) ** 2 / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        results = numerator / denominator

        for i in range(len(results)):
            if results[i] == 0:
                results[i] = reg_param

        return results

class QuadraticDiscriminantAnalysis:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = {}
        self.covariance = {}
        self.priors = {}

        for cls in self.classes:
            X_c = X[y == cls]
            self.mean[cls] = np.mean(X_c, axis=0)
            self.covariance[cls] = np.cov(X_c, rowvar=False) + np.eye(X.
 ↪shape[1]) * reg_param
            self.priors[cls] = X_c.shape[0] / X.shape[0]

        #print("Means:", self.mean)
        #print("Covariances:", self.covariance)
        #print("Priors:", self.priors)
```

```
    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        discriminants = []

        for cls in self.classes:
            G = self._quadratic_discriminant(cls, x)
            discriminants.append(G)

        return self.classes[np.argmax(discriminants)]

    def _quadratic_discriminant(self, cls, x):
        mean = self.mean[cls]
        covariance = self.covariance[cls]
        inv_covariance = np.linalg.inv(covariance)
        det_cov = np.linalg.det(covariance)

        if det_cov == 0:
            det_cov = reg_param

        W_k = -0.5 * inv_covariance
        w_k = np.dot(inv_covariance, mean)
        w_k0 = -0.5 * np.dot(mean.T, np.dot(inv_covariance, mean)) - 0.5 * np.
↪log(det_cov) + np.log(self.priors[cls])

        discriminant = np.dot(x.T, np.dot(W_k, x)) + np.dot(w_k.T, x) + w_k0

        return discriminant

class LinearDiscriminantAnalysis:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = {}
        self.covariance = {}
        self.priors = {}

        for cls in self.classes:
            X_c = X[y == cls]
            self.mean[cls] = np.mean(X_c, axis=0)
            self.covariance[cls] = np.cov(X_c, rowvar=False) + np.eye(X.
↪shape[1]) * reg_param
            self.priors[cls] = X_c.shape[0] / X.shape[0]

        #print("Means:", self.mean)
```

```
        #print("Covariances:", self.covariance)
        #print("Priors:", self.priors)

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        discriminants = []

        for cls in self.classes:
            G = self._linear_discriminant(cls, x)
            discriminants.append(G)

        return self.classes[np.argmax(discriminants)]

    def _linear_discriminant(self, cls, x):
        mean = self.mean[cls]
        covariance = self.covariance[cls]
        inv_covariance = np.linalg.inv(covariance)

        w_k = np.dot(inv_covariance, mean)
        w_k0 = -0.5 * np.dot(mean.T, np.dot(inv_covariance, mean)) + np.
 ↪log(self.priors[cls])

        discriminant = np.dot(w_k.T, x) + w_k0

        return discriminant

def confusion_matrix(y_true, y_pred, num_classes):
    cm = np.zeros((num_classes, num_classes), dtype=int)
    for i in range(len(y_true)):
        cm[y_true[i]][y_pred[i]] += 1
    return cm

def accuracy_per_class(cm):
    return np.diag(cm) / np.sum(cm, axis=1)

def overall_accuracy(cm):
    return np.sum(np.diag(cm)) / np.sum(cm)

data, labels = load_data_from_csv(case1_csv_filename)

X_train, X_test, y_train, y_test = train_test_split(data, labels)

# Standardizing the features
scaler = StandardScaler()
```

```python
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

nb = NaiveBayesClassifier()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
cm_nb = confusion_matrix(y_test, y_pred_nb, len(classes))
acc_nb = accuracy_per_class(cm_nb)
overall_acc_nb = overall_accuracy(cm_nb)

qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)
y_pred_qda = qda.predict(X_test)
cm_qda = confusion_matrix(y_test, y_pred_qda, len(classes))
acc_qda = accuracy_per_class(cm_qda)
overall_acc_qda = overall_accuracy(cm_qda)

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
y_pred_lda = lda.predict(X_test)
cm_lda = confusion_matrix(y_test, y_pred_lda, len(classes))
acc_lda = accuracy_per_class(cm_lda)
overall_acc_lda = overall_accuracy(cm_lda)

print("\n--------------------CASE1--------------------\n")

print("Bayes Confusion Matrix:\n", cm_nb)
print("Bayes Accuracy per Class:\n", acc_nb)
print("Bayes Accuracy:", overall_acc_nb)

print("\n----------------------------------------\n")

print("QDA Confusion Matrix:\n", cm_qda)
print("QDA Accuracy per Class:\n", acc_qda)
print("QDA Accuracy:", overall_acc_qda)

print("\n----------------------------------------\n")

print("LDA Confusion Matrix:\n", cm_lda)
print("LDA Accuracy per Class:\n", acc_lda)
print("LDA Accuracy:", overall_acc_lda)

data, labels = load_data_from_csv(case2_csv_filename)

X_train, X_test, y_train, y_test = train_test_split(data, labels)

# Standardizing the features
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

nb = NaiveBayesClassifier()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
cm_nb = confusion_matrix(y_test, y_pred_nb, len(classes))
acc_nb = accuracy_per_class(cm_nb)
overall_acc_nb = overall_accuracy(cm_nb)

qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)
y_pred_qda = qda.predict(X_test)
cm_qda = confusion_matrix(y_test, y_pred_qda, len(classes))
acc_qda = accuracy_per_class(cm_qda)
overall_acc_qda = overall_accuracy(cm_qda)

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
y_pred_lda = lda.predict(X_test)
cm_lda = confusion_matrix(y_test, y_pred_lda, len(classes))
acc_lda = accuracy_per_class(cm_lda)
overall_acc_lda = overall_accuracy(cm_lda)

print("\n--------------------CASE2--------------------\n")

print("Bayes Confusion Matrix:\n", cm_nb)
print("Bayes Accuracy per Class:\n", acc_nb)
print("Bayes Accuracy:", overall_acc_nb)

print("\n----------------------------------------\n")

print("QDA Confusion Matrix:\n", cm_qda)
print("QDA Accuracy per Class:\n", acc_qda)
print("QDA Accuracy:", overall_acc_qda)

print("\n----------------------------------------\n")

print("LDA Confusion Matrix:\n", cm_lda)
print("LDA Accuracy per Class:\n", acc_lda)
print("LDA Accuracy:", overall_acc_lda)
```