**2024年秋季学习课后作业**

第1次作业

1. P42-P43 1 5 6
2. 计算点（1，2，3）和点（3，4，8）的L1、L2、L∞距离。

L1 距离（曼哈顿距离）：

L2 距离（欧几里得距离）：

L∞ 距离（切比雪夫距离）：

1. 试编写程序，利用本章提供的k-means算法代码或者sklearn.cluster.KMeans算法函数实现二分k-means算法，对随书资源中的kmeansSamples.txt文件中的点进行分簇，并于k-means算法的效果进行比较。

代码:

# -\*- coding: utf-8 -\*-

'''

@File    :   2.5.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import numpy as np

import matplotlib.pyplot as plt

def L2(vecXi, vecXj):

    '''计算欧氏距离'''

    return np.sqrt(np.sum(np.power(vecXi - vecXj, 2)))

def kMeans(S, k, distMeas=L2):

    '''K均值聚类'''

    m = np.shape(S)[0]  # 样本总数

    sampleTag = np.zeros(m)

    # 随机产生k个初始簇中心

    n = np.shape(S)[1]  # 样本向量的特征数

    clusterCents = np.mat([[-1.93964824, 2.33260803], [7.79822795, 6.72621783], [10.64183154, 0.20088133]])

    sampleTagChanged = True

    SSE = 0.0

    while sampleTagChanged:

        sampleTagChanged = False

        SSE = 0.0

        # 计算每个样本点到各簇中心的距离

        for i in range(m):

            minD = np.inf

            minIndex = -1

            for j in range(len(clusterCents)):

                d = distMeas(clusterCents[j, :], S[i, :])

                if d < minD:

                    minD = d

                    minIndex = j

            if sampleTag[i] != minIndex:

                sampleTagChanged = True

            sampleTag[i] = minIndex

            SSE += minD \*\* 2

        # 重新计算簇中心

        for i in range(len(clusterCents)):

            ClustI = S[np.nonzero(sampleTag[:] == i)[0]]

            if len(ClustI) > 0:

                clusterCents[i, :] = np.mean(ClustI, axis=0)

    return clusterCents, sampleTag, SSE

def bisectingKMeans(S, k):

    '''二分K均值聚类'''

    clusters = [S]  # 初始只有一个簇

    clusterCents = []

    while len(clusters) < k:

        # 找到拥有最多样本的簇

        maxClusterIndex = np.argmax([len(cluster) for cluster in clusters])

        currentCluster = clusters[maxClusterIndex]

        # 对当前簇进行K-means聚类分成2个簇

        newCents, newTags, \_ = kMeans(currentCluster, 2)

        # 更新簇列表，用新的两个簇替换旧簇

        clusters.pop(maxClusterIndex)  # 移除旧的簇

        clusters.append(currentCluster[newTags == 0])  # 添加第一个新簇

        clusters.append(currentCluster[newTags == 1])  # 添加第二个新簇

        # 记录新的簇中心

        clusterCents.append(newCents)

    # 最终簇中心

    finalCenters = []

    for cluster in clusters:

        center = np.mean(cluster, axis=0)

        finalCenters.append(center)

    return np.array(finalCenters), clusters

if \_\_name\_\_ == '\_\_main\_\_':

    samples = np.loadtxt("kmeansSamples.txt")

    clusterCents, clusters = bisectingKMeans(samples, 3)

    # 可视化结果

    plt.figure()

    for i, cluster in enumerate(clusters):

        plt.scatter(cluster[:, 0], cluster[:, 1], label=f"Cluster {i + 1}")

    plt.scatter(clusterCents[:, 0], clusterCents[:, 1], c='red', marker='^', s=100, label='Centroids')

    plt.title('Bisecting K-means Clustering')

    plt.legend()

    plt.show()

    print("Final Cluster Centers:")

    print(clusterCents)

结果:

图形用户界面

描述已自动生成

6、在k-means算法中，当采用曼哈顿距离作为距离度量方法、以距离和作为损失函数时，称为k-median算法。

1. 簇中心如何计算？
2. 修改本章提供的k-means算法代码实现该算法。
3. # -\*- coding: utf-8 -\*-
4. '''
5. @File    :   2.6.py
6. @Time    :   2024/11/24 14:00:00
7. @Author  :   210162401014 靳琛飞
8. '''
9. import numpy as np
10. import matplotlib.pyplot as plt
11. def L1(vecXi, vecXj):
12. return np.sum(np.abs(vecXi - vecXj))
13. def kMedian(S, k, distMeas=L1):
14. m = np.shape(S)[0]
15. sampleTag = np.zeros(m)
16. n = np.shape(S)[1]
17. clusterCents = np.mat([[-1.93964824, 2.33260803], [7.79822795, 6.72621783], [10.64183154, 0.20088133]])
18. sampleTagChanged = True
19. SSE = 0.0
20. while sampleTagChanged:
21. sampleTagChanged = False
22. SSE = 0.0
23. for i in range(m):
24. minD = np.inf
25. minIndex = -1
26. for j in range(k):
27. d = distMeas(clusterCents[j, :], S[i, :])
28. if d < minD:
29. minD = d
30. minIndex = j
31. if sampleTag[i] != minIndex:
32. sampleTagChanged = True
33. sampleTag[i] = minIndex
34. SSE += minD
35. for i in range(k):
36. ClustI = S[np.nonzero(sampleTag[:] == i)[0]]
37. clusterCents[i, :] = np.median(ClustI, axis=0)
38. return clusterCents, sampleTag, SSE
39. if \_\_name\_\_ == '\_\_main\_\_':
40. samples = np.loadtxt("kmeansSamples.txt")
41. clusterCents, sampleTag, SSE = kMedian(samples, 3)
42. plt.scatter(clusterCents[:, 0].tolist(), clusterCents[:, 1].tolist(), c='r', marker='^')
43. plt.scatter(samples[:, 0], samples[:, 1], c=sampleTag, linewidths=np.power(sampleTag + 0.5, 2))
44. plt.title('K-median Clustering')
45. plt.show()
46. print("Cluster Centers:")
47. print(clusterCents)
48. print("Sum of Manhattan Distances:")
49. print(SSE)
50. P78-P79 1 3 4
51. 用sklearn.linear\_model包中的LinearRegression对表3-1所示的示例进行线性回归实验，比较结果。
52. # -\*- coding: utf-8 -\*-
53. '''
54. @File    :   3.1.py
55. @Time    :   2024/11/24 14:00:00
56. @Author  :   210162401014 靳琛飞
57. '''
58. from sklearn import linear\_model  # 导入线性模型
59. import matplotlib.pyplot as plt    # 用于作图
60. import numpy as np                 # 用于创建向量
61. # 创建线性回归模型
62. reg = linear\_model.LinearRegression(fit\_intercept=True, normalize=False)
63. # 输入数据
64. x = [[15], [20], [25], [30], [35], [40]]
65. y = [136, 140, 155, 160, 157, 175]
66. # 训练模型
67. reg.fit(x, y)
68. # 获取模型参数
69. k = reg.coef\_[0]  # 获取斜率
70. b = reg.intercept\_  # 获取截距
71. # 生成预测值并可视化
72. x0 = np.arange(15, 41)  # 生成从15到40的数值，包括40
73. y0 = k \* x0 + b  # 计算预测值
74. plt.scatter(x, y, color='blue', label='Data Points')  # 绘制散点图
75. plt.plot(x0, y0, color='red', label='Regression Line')  # 绘制线性回归直线
76. plt.xlabel('Temperature')
77. plt.ylabel('Measured Value')
78. plt.title('Linear Regression Fit')
79. plt.legend()
80. plt.grid()
81. plt.show()
82. # 打印斜率和截距
83. print("k =", k, "b =", b, "\n")
84. # 对新的温度值进行预测
85. new\_tempera = [18]
86. new\_tempera = (np.mat(new\_tempera)).T
87. pro\_num = k \* new\_tempera + b
88. print("Predicted value for temperature 18:", pro\_num[0, 0])
89. new\_tempera = [22]
90. new\_tempera = (np.mat(new\_tempera)).T
91. pro\_num = k \* new\_tempera + b
92. print("Predicted value for temperature 22:", pro\_num[0, 0])
93. new\_tempera = [33]
94. new\_tempera = (np.mat(new\_tempera)).T
95. pro\_num = k \* new\_tempera + b
96. print("Predicted value for temperature 33:", pro\_num[0, 0])
97. 查阅资料，研究梯度下降法中步长的动态调整方法，试将代码3-6中固定步长改为动态步长，并对比两者运行结果。
98. '''
99. @File    :   3.3.py
100. @Time    :   2024/11/24 14:00:00
101. @Author  :   210162401014 靳琛飞
102. '''
103. import numpy as np
104. def lossValue(X, y, W):
105. return np.sum(np.square(X \* W - y)) / (2 \* len(y))
106. def gradient(X, y, W):
107. return X.T \* (X \* W - y) / len(y)
108. temperatures = [15, 20, 25, 30, 35, 40]
109. flowers = [136, 140, 155, 160, 157, 175]
110. X = (np.mat([[1, 1, 1, 1, 1, 1], temperatures])).T
111. y = (np.mat(flowers)).T
112. W = (np.mat([0.0, 0.0])).T
113. print(W)
114. alpha = 0.00025
115. loss\_change = 0.000001
116. loss = lossValue(X, y, W)
117. for i in range(30000):
118. grad = gradient(X, y, W)
119. alpha = 0.00025 / (1 + i)  # 动态调整步长
120. W = W - alpha \* grad
121. newloss = lossValue(X, y, W)
123. print(str(i) + ": " + str(W[0]) + ' : ' + str(W[1]))
124. print(newloss)
126. if abs(loss - newloss) < loss\_change:
127. break
129. loss = newloss
130. new\_tempera = [18, 22, 33]
131. new\_tempera = (np.mat([[1, 1, 1], new\_tempera])).T
132. pro\_num = new\_tempera \* W
133. print(pro\_num)

4、试修改代码3-6实现批梯度下降和随机梯度下降算法，并从时间和结果两方面与原算法进行比较。

'''

@File    :   3.4.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import numpy as np

import time

def lossValue(X, y, W):

    return np.sum(np.square(X \* W - y)) / (2 \* len(y))

def gradient(X, y, W):

    return X.T \* (X \* W - y) / len(y)

def batch\_gradient\_descent(X, y, W, alpha, loss\_change, max\_iter):

    loss = lossValue(X, y, W)

    for i in range(max\_iter):

        W = W - alpha \* gradient(X, y, W)

        newloss = lossValue(X, y, W)

        if abs(loss - newloss) < loss\_change:

            break

        loss = newloss

    return W

def stochastic\_gradient\_descent(X, y, W, alpha, loss\_change, max\_iter):

    m = len(y)

    loss = lossValue(X, y, W)

    for i in range(max\_iter):

        for j in range(m):

            rand\_ind = np.random.randint(0, m)

            x\_i = X[rand\_ind, :]

            y\_i = y[rand\_ind, :]

            grad = x\_i.T \* (x\_i \* W - y\_i)

            W = W - alpha \* grad

        newloss = lossValue(X, y, W)

        if abs(loss - newloss) < loss\_change:

            break

        loss = newloss

    return W

temperatures = [15, 20, 25, 30, 35, 40]

flowers = [136, 140, 155, 160, 157, 175]

X = np.mat([[1]\*len(temperatures), temperatures]).T

y = np.mat(flowers).T

W = np.mat([0.0, 0.0]).T

# Batch Gradient Descent

start\_time = time.time()

alpha = 0.00025

loss\_change = 0.000001

max\_iter = 30000

W\_batch = batch\_gradient\_descent(X, y, W, alpha, loss\_change, max\_iter)

end\_time = time.time()

print("Batch Gradient Descent:")

print("Weights:", W\_batch)

print("Time taken:", end\_time - start\_time, "seconds")

# Stochastic Gradient Descent

start\_time = time.time()

alpha = 0.00025

loss\_change = 0.000001

max\_iter = 3000

W\_stochastic = stochastic\_gradient\_descent(X, y, W, alpha, loss\_change, max\_iter)

end\_time = time.time()

print("\nStochastic Gradient Descent:")

print("Weights:", W\_stochastic)

print("Time taken:", end\_time - start\_time, "seconds")

第2次作业

1. P131 1 5 6 7
2. 将表4-1所示的样本集合按年龄（大于等于29）进行切分，试计算切分后的信息增益和基尼指数。
3. # -\*- coding: utf-8 -\*-
4. '''
5. @File    :   4.1.py
6. @Time    :   2024/11/24 14:00:00
7. @Author  :   210162401014 靳琛飞
8. '''
9. import pandas as pd
10. import numpy as np
11. # 定义计算熵的函数
12. def entropy(data):
13. classes, counts = np.unique(data, return\_counts=True)
14. probabilities = counts / len(data)
15. return -np.sum(probabilities \* np.log2(probabilities))
16. # 定义计算基尼指数的函数
17. def gini(data):
18. classes, counts = np.unique(data, return\_counts=True)
19. probabilities = counts / len(data)
20. return 1 - np.sum(probabilities \*\* 2)
21. # 创建示例数据表格
22. data = {
23. '编号': [1, 2, 3, 4, 5],
24. '年龄（岁）': [35,38,26,29,28],
25. '身高（cm）': [176, 178, 172, 173, 174],
26. '学历': ['本科', '硕士', '本科', '博士',  '本科'],
27. '月薪（元）': [20000, 10000, 25000, 20000, 15000],
28. '是否相亲': ['否', '是', '否', '是', '是']
29. }
30. df = pd.DataFrame(data)
31. # 按照年龄（大于等于29岁）切分数据集
32. left\_subset = df[df['年龄（岁）'] >= 29]['是否相亲']
33. right\_subset = df[df['年龄（岁）'] < 29]['是否相亲']
34. # 计算切分后的信息增益
35. total\_entropy = entropy(df['是否相亲'])
36. left\_entropy = entropy(left\_subset)
37. right\_entropy = entropy(right\_subset)
38. weighted\_entropy = (len(left\_subset) / len(df)) \* left\_entropy + (len(right\_subset) / len(df)) \* right\_entropy
39. information\_gain = total\_entropy - weighted\_entropy
40. # 计算切分后的基尼指数
41. gini\_left = gini(left\_subset)
42. gini\_right = gini(right\_subset)
43. weighted\_gini = (len(left\_subset) / len(df)) \* gini\_left + (len(right\_subset) / len(df)) \* gini\_right
44. print("I信息增益:", information\_gain)
45. print("基尼指数:", weighted\_gini)
46. 表4-6为某二分类器预测结果的混淆矩阵，试计算准确率、平均准确率、精确率、召回率和F1—score。

表 4-6 某二分类器预测结果

|  |  |  |
| --- | --- | --- |
|  | 预测为“0”的样本数 | 预测为“1”的样本数 |
| 标签为“0”的样本数 | 1026 | 1101 |
| 标签为“1”的样本数 | 1007 | 911026 |

# -\*- coding: utf-8 -\*-

'''

@File    :   4.5.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

# 导入必要的库（如果需要）

import numpy as np

# 混淆矩阵数据

TP = 1026  # 真正例

FP = 1101  # 假正例

TN = 911026  # 真负例

FN = 1007  # 假负例

# 计算准确率

accuracy = (TP + TN) / (TP + FP + TN + FN)

# 计算精确率

precision = TP / (TP + FP) if (TP + FP) > 0 else 0

# 计算召回率

recall = TP / (TP + FN) if (TP + FN) > 0 else 0

# 计算F1-score

f1\_score = (2 \* precision \* recall) / (precision + recall) if (precision + recall) > 0 else 0

# 输出结果

print("准确率: {:.4f}".format(accuracy))

print("精确率: {:.4f}".format(precision))

print("召回率: {:.4f}".format(recall))

print("F1-score: {:.4f}".format(f1\_score))

1. 针对随书资源ellipseSamples.txt文件中的二分类样本点，设计逻辑回归模型，进行训练，画出决策边界。
2. # -\*- coding: utf-8 -\*-
3. '''
4. @File    :   4.6.py
5. @Time    :   2024/11/24 14:00:00
6. @Author  :   210162401014 靳琛飞
7. '''
8. import numpy as np
9. import pandas as pd
10. import matplotlib.pyplot as plt
11. from sklearn.linear\_model import LogisticRegression
12. # 读取数据
13. data = pd.read\_csv('ellipseSamples.txt', delim\_whitespace=True, header=None)
14. X = data.iloc[:, :-1].values  # 特征
15. y = data.iloc[:, -1].values    # 标签
16. # 创建逻辑回归模型
17. model = LogisticRegression()
18. # 训练模型
19. model.fit(X, y)
20. # 创建网格以评估模型
21. x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1
22. y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1
23. # 增加步长以减少网格大小
24. step\_size = 5.0  # 更大的步长，例如 5.0
25. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, step\_size), np.arange(y\_min, y\_max, step\_size))
26. # 预测每个网格点的类
27. Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])
28. Z = Z.reshape(xx.shape)
29. # 绘制决策边界
30. plt.contourf(xx, yy, Z, alpha=0.8)
31. plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')
32. plt.title('Logistic Regression Decision Boundary')
33. plt.xlabel('Feature 1')
34. plt.ylabel('Feature 2')
35. plt.show()
36. 查阅资料学习sklearn.ensemble.ExtraTreesClassifier的应用方法，并用它来完成优惠券使用示例实验。
37. # -\*- coding: utf-8 -\*-
38. '''
39. @File    :   4.9.py
40. @Time    :   2024/11/24 14:00:00
41. @Author  :   210162401014 靳琛飞
42. '''
43. import numpy as np
44. import pandas as pd
45. import time
46. import os
47. import uuid
48. ### 0.定义函数及公用变量、文件等
49. file\_print\_to = open("file\_print\_to1.txt", 'a')
50. print('\n\n\n--\*\*\*\*-- 本次实验开始时间：' + time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time())), file=file\_print\_to)
51. print('-------- 实验计算机名：' + str(os.environ['COMPUTERNAME'] \
52. + '   MAC地址：' + str(uuid.UUID(int = uuid.getnode()).hex[-12:])), file=file\_print\_to)
53. data\_file = 'feature\_offline\_in15days\_2018\_01\_06.csv'
54. print('-------- 实验数据文件是：' + data\_file, file=file\_print\_to)
55. ### 1.读取数据，划分训练集（train）和验证集（verify）
56. print('1.读取数据，划分训练集（train）和验证集（verify）')
57. features = pd.read\_csv(data\_file)
58. features\_train = features[features.Date\_received <= 20160501]
59. features\_verify = features[(features.Date\_received >= 20160516) & \
60. (features.Date\_received <= 20160616)]
61. ### 2.指定训练用的特征，生成训练特征集（X\_train）、训练标签集（y\_train）、验证特征集（X\_verify）、验证标签集（y\_verify）。
62. print('2.指定训练用的特征，生成训练特征集（X\_train）、训练标签集（y\_train）、验证特征集（X\_verify）、验证标签集（y\_verify）')
63. fe\_parameters = ['User\_id', 'Merchant\_id', \
64. #-------- 用户、优惠券相关特征 --------
65. 'distance', 'discount\_man', 'discount\_jian', 'discount\_rate', \
66. 'day\_of\_week', 'is\_weekend', 'day\_of\_month', \
67. #-------- 商户相关特征 --------
68. 'total\_sales', 'sales\_use\_coupon', 'total\_coupons', \
69. 'use\_coupon\_rate', 'transfer\_rate', 'merchant\_max\_distance', \
70. 'merchant\_min\_distance', 'merchant\_mean\_distance' \
71. ]
72. print('采用' + str(len(fe\_parameters)) + '个特征：' +  ','.join(fe\_parameters))
73. print('采用' + str(len(fe\_parameters)) + '个特征：' +  ','.join(fe\_parameters), file=file\_print\_to)
74. X\_train = features\_train[fe\_parameters]
75. y\_train = np.ravel(features\_train[['coupon\_apply']])
76. X\_verify = features\_verify[fe\_parameters]
77. y\_verify = np.ravel(features\_verify[['coupon\_apply']])
78. ### 3.应用ExtraTreesClassifier算法
79. print('3.应用ExtraTreesClassifier算法')
80. print('\n\n------ExtraTreesClassifier预测', file=file\_print\_to)
81. time\_start = time.time()
82. from sklearn.ensemble import ExtraTreesClassifier
83. etc = ExtraTreesClassifier(random\_state=2)
84. etc.fit(X\_train, y\_train)
85. print('训练用时：' + str(time.time() - time\_start), file=file\_print\_to)
86. print('采用ExtraTreesClassifier预测的准确率：' + str(etc.score(X\_verify, y\_verify)), file=file\_print\_to)
87. print('\n各特征重要程度：', file=file\_print\_to)
88. print('\n各特征重要程度：', file=file\_print\_to)
89. print(list(zip(fe\_parameters, map(lambda x: round(x, 4), etc.feature\_importances\_))), file=file\_print\_to)
90. file\_print\_to.close()

第3次作业

1. P151 1 3
2. 用马赛克图分析优惠券核销示例中的周一到周日的领券核销情况。

# -\*- coding: utf-8 -\*-

'''

@File    :   5.1.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import pandas as pd

from statsmodels.graphics.mosaicplot import mosaic

import matplotlib.pyplot as plt

# 读取数据集

df = pd.read\_csv(r'feature\_offline\_in15days\_2018\_01\_06.csv')

# 输出 DataFrame 的前几行和列名，检查是否有日期列

print(df.head())  # 打印前几行数据

print(df.columns)  # 打印所有列名

# 使用 'Date' 列并将其转换为 datetime 类型

# 首先处理可能的 NaN 或格式问题，确保 Date 列是字符串类型

df['Date'] = df['Date'].astype(str).str.replace('.0', '', regex=False)

# 然后将其转换为 datetime 格式

df['Date'] = pd.to\_datetime(df['Date'], format='%Y%m%d', errors='coerce')  # 使用合适的日期格式

# 检查转换后的结果

print(df['Date'].head())

# 提取星期几

df['weekday'] = df['Date'].dt.day\_name()

# 确保 coupon\_apply 列存在并进行处理

if 'coupon\_apply' in df.columns:

    label = df['coupon\_apply'].astype('str').apply(lambda x: '用券' if x == '1' else '不用')

else:

    print("数据集中找不到 'coupon\_apply' 列，请检查列名")

# 检查是否成功创建了 weekday 列

if 'weekday' in df.columns:

    # 创建一个 DataFrame 用于马赛克图

    mosaic\_data = pd.concat([df['weekday'], label], axis=1)

    mosaic\_data.columns = ['weekday', 'coupon\_apply']  # 重命名列

    # 绘制马赛克图

    plt.figure(figsize=(12, 6))

    mosaic(data=mosaic\_data, index=['weekday', 'coupon\_apply'], gap=0.01, title='星期几与使用优惠券的关系')

    plt.xticks(rotation=45)  # 可以旋转X轴标签以提高可读性

    plt.show()

else:

    print("未能创建 'weekday' 列，无法绘制马赛克图。")

1. 探索sklearn的decomposition包中的PCA类的应用。如用它对本章两个降维示例进行降维处理。

# -\*- coding: utf-8 -\*-

'''

@File    :   5.3.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

from sklearn.decomposition import PCA

# 将原始矩阵转换为二维数组

data = A.A  # 使用 .A 将矩阵转换为 NumPy 数组

print("\nUsing PCA for dimensionality reduction:")

# 初始化 PCA，指定降维后的维度

pca = PCA(n\_components=1)  # 降维到 1 维

data\_reduced = pca.fit\_transform(data)

# 打印降维后的结果

print("Reduced data:")

print(data\_reduced)

# 打印主成分

print("Principal components:")

print(pca.components\_)

# 还原数据

reconstructed\_data = pca.inverse\_transform(data\_reduced)

print("Reconstructed data from reduced dimensions:")

print(reconstructed\_data)

1. P185 4 5 6
2. 在6.5.2节的示例中，用前向算法计算观测序列10、11、7的概率。

# -\*- coding: utf-8 -\*-

'''

@File    :   6.4.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

def forward\_algorithm(pi, A, B, observations):

    num\_states = len(pi)

    num\_observations = len(observations)

    # 初始化前向概率矩阵

    alpha = np.zeros((num\_observations, num\_states))

    # 初始时刻 t=0

    first\_observation = observations[0]

    for j in range(num\_states):

        alpha[0][j] = pi[j] \* B[j][first\_observation - 2]  # -2 因为 observations 是从 2 开始的

    # 递推计算

    for t in range(1, num\_observations):

        current\_observation = observations[t]

        for j in range(num\_states):

            alpha[t][j] = sum(alpha[t-1][i] \* A[i][j] for i in range(num\_states)) \* B[j][current\_observation - 2]

    # 计算最终概率

    probability = sum(alpha[num\_observations - 1][j] for j in range(num\_states))

    return probability

# 测试观测序列 [10, 11, 7]

observations\_sequence = [10, 11, 7]

probability = forward\_algorithm(pi, A, B, observations\_sequence)

print("观测序列 [10, 11, 7] 的概率:", probability)

1. 在6.5.2节的示例中，用维特比算法计算观测序列10、11、7时最大可能的状态序列。

# -\*- coding: utf-8 -\*-

'''

@File    :   6.5.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import numpy as np

def viterbi\_algorithm(pi, A, B, observations):

    num\_states = len(pi)

    num\_observations = len(observations)

    # 初始化 delta 和 psi 矩阵

    delta = np.zeros((num\_observations, num\_states))

    psi = np.zeros((num\_observations, num\_states), dtype=int)

    # 第一步初始化

    first\_observation = observations[0]

    for j in range(num\_states):

        delta[0][j] = pi[j] \* B[j][first\_observation - 2]  # 观测序列从2开始

    # 动态规划计算

    for t in range(1, num\_observations):

        current\_observation = observations[t]

        for j in range(num\_states):

            max\_prob, max\_state = max((delta[t-1][i] \* A[i][j], i) for i in range(num\_states))

            delta[t][j] = max\_prob \* B[j][current\_observation - 2]

            psi[t][j] = max\_state  # 记录最优前驱状态

    # 找到最终状态序列

    best\_path = np.zeros(num\_observations, dtype=int)

    best\_last\_state = np.argmax(delta[num\_observations - 1])

    best\_path[num\_observations - 1] = best\_last\_state

    # 回溯得到最优状态序列

    for t in range(num\_observations - 2, -1, -1):

        best\_path[t] = psi[t + 1][best\_path[t + 1]]

    return best\_path, delta

# 测试观测序列 [10, 11, 7]

observations\_sequence = [10, 11, 7]

best\_state\_sequence, delta\_matrix = viterbi\_algorithm(pi, A, B, observations\_sequence)

# 输出结果

print("观测序列 [10, 11, 7] 的最可能状态序列:", best\_state\_sequence)

print("Delta 矩阵:\n", delta\_matrix)

1. 设计一个隐马尔可夫模型的例子，并尝试用随书资源提供的程序去计算概率、预测状态序列。

# -\*- coding: utf-8 -\*-

'''

@File    :   6.6.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import numpy as np

# 隐藏状态集合

states = ['Sunny', 'Rainy']

# 初始状态概率向量

pi = np.array([0.6, 0.4])

# 状态转移矩阵 A

A = np.array([[0.8, 0.2],

              [0.4, 0.6]])

# 观测概率矩阵 B

B = np.array([[0.4, 0.6],

              [0.7, 0.3]])

# 前向算法

def forward\_algorithm(pi, A, B, observations):

    num\_states = len(pi)

    num\_observations = len(observations)

    alpha = np.zeros((num\_observations, num\_states))

    # 初始化第一步

    for j in range(num\_states):

        alpha[0][j] = pi[j] \* B[j][observations[0]]

    # 递推计算

    for t in range(1, num\_observations):

        for j in range(num\_states):

            alpha[t][j] = np.sum(alpha[t-1] \* A[:,j]) \* B[j][observations[t]]

    probability = np.sum(alpha[num\_observations-1])

    return probability

# 维特比算法

def viterbi\_algorithm(pi, A, B, observations):

    num\_states = len(pi)

    num\_observations = len(observations)

    delta = np.zeros((num\_observations, num\_states))

    psi = np.zeros((num\_observations, num\_states), dtype=int)

    # 初始化第一步

    for j in range(num\_states):

        delta[0][j] = pi[j] \* B[j][observations[0]]

    # 递推计算

    for t in range(1, num\_observations):

        for j in range(num\_states):

            max\_prob, max\_state = max((delta[t-1][i] \* A[i][j], i) for i in range(num\_states))

            delta[t][j] = max\_prob \* B[j][observations[t]]

            psi[t][j] = max\_state

    # 回溯得到最优状态序列

    best\_path = np.zeros(num\_observations, dtype=int)

    best\_last\_state = np.argmax(delta[num\_observations - 1])

    best\_path[num\_observations - 1] = best\_last\_state

    for t in range(num\_observations - 2, -1, -1):

        best\_path[t] = psi[t + 1][best\_path[t + 1]]

    return best\_path

# 测试观测序列

observations\_sequence = [0, 1, 1]  # 表示观测到 Coat 和 T-shirt

# 计算概率

probability\_forward = forward\_algorithm(pi, A, B, observations\_sequence)

print("观测序列的概率(前向算法):", probability\_forward)

# 预测状态序列

best\_state\_sequence = viterbi\_algorithm(pi, A, B, observations\_sequence)

print("预测的最可能状态序列:", [states[i] for i in best\_state\_sequence])

第4次作业

1. P219-220 1 2 3 5 7
2. 编写程序实现感知机学习算法。可用随书资源“平面二分类线性逻辑回归示例.ipynb”程序文件中的方法产生100个实验点，用来训练感知机学习算法。

# -\*- coding: utf-8 -\*-

'''

@File    :   7.1.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import numpy as np

# 初始化权重和偏置

w = np.random.rand(2)

b = np.random.rand()

# 感知机学习算法

def perceptron\_learning(samples, epochs=100, learning\_rate=0.1):

    global w, b  # 声明w和b为全局变量

    for epoch in range(epochs):

        for sample in samples:

            x = np.array(sample[:2])

            y = sample[2]

            y\_pred = 1 if np.dot(w, x) + b > 0 else 0

            w += learning\_rate \* (y - y\_pred) \* x

            b += learning\_rate \* (y - y\_pred)

    return w, b

# 将感知机学习算法应用于生成的样本

weights, bias = perceptron\_learning(samples)

print("学到的权重:", weights)

print("学到的偏置:", bias)

1. 在7.2.1节的三层感知机的误差反向传播学习示例中，计算第2个训练样本（0，1）的前向传播过程。网络参数的初值与示例初值相同：W1=，θ1=，W2=,θ2=。

# -\*- coding: utf-8 -\*-

'''

@File    :   7.2.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import numpy as np

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

# 定义权重和阈值

W1 = np.array([[0.1, 0.2],

               [0.2, 0.3]])  # 第1隐层的连接权重系数

theta1 = np.array([0.3, 0.3])  # 第1隐层的阈值

W2 = np.array([[0.4, 0.5],

               [0.4, 0.5]])  # 第2隐层的连接权重系数

theta2 = np.array([0.6, 0.6])  # 第2隐层的阈值

# 输入样本

X = np.array([0.0, 1.0])

# 计算第1隐层节点1的输出

def y\_1\_1(W1, theta1, X):

    return sigmoid(W1[0, 0] \* X[0] + W1[1, 0] \* X[1] + theta1[0])

# 计算第1隐层节点2的输出

def y\_1\_2(W1, theta1, X):

    return sigmoid(W1[0, 1] \* X[0] + W1[1, 1] \* X[1] + theta1[1])

# 计算第2隐层节点1的输出

def y\_2\_1(W2, theta2, Y1):

    return sigmoid(W2[0, 0] \* Y1[0] + W2[1, 0] \* Y1[1] + theta2[0])

# 计算第2隐层节点2的输出

def y\_2\_2(W2, theta2, Y1):

    return sigmoid(W2[0, 1] \* Y1[0] + W2[1, 1] \* Y1[1] + theta2[1])

# 前向传播计算

Y1 = np.zeros(2)

Y2 = np.zeros(2)

Y1[0] = y\_1\_1(W1, theta1, X)

Y1[1] = y\_1\_2(W1, theta1, X)

Y2[0] = y\_2\_1(W2, theta2, Y1)

Y2[1] = y\_2\_2(W2, theta2, Y1)

print("第2隐层节点1的输出：", Y2[0])

print("第2隐层节点2的输出：", Y2[1])

1. 在第2题条件下，计算反向传播学习过程中ω1（1，2）的更新。

# -\*- coding: utf-8 -\*-

'''

@File    :   7.3.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

# 定义目标输出

D = np.array([0.7, 0.9])

# 计算输出层的误差项

E2 = D - Y2

# 计算第2隐层的误差项

E1 = np.dot(W2.T, E2) \* Y1 \* (1 - Y1)

# 更新W1[1,2]

a = 0.1  # 学习率

W1[0, 1] = W1[0, 1] + a \* E1[1] \* Y1[1] \* (1 - Y1[1]) \* X[0]

W1[1, 1] = W1[1, 1] + a \* E1[1] \* Y1[1] \* (1 - Y1[1]) \* X[1]

print("更新后的W1矩阵:")

print(W1)

5、基于随书提供的源程序NN\_regress.py，修改网络结构和参数，重现图7-11~图7-13的拟合结果。

7、对代码7-3所示的示例，分别应用交叉熵、相对熵、余弦相似度和双曲余弦对数等损失函数，比较它们的效果和训练时长。

第5次作业

1. P240 1 2
2. 与MNIST手写体数字集一样，CIFAR-10包含了60000张图片，共10类。训练集50000张，测试集10000张。但与MNIST不同的是，CIFAR-10数据集中的图片是彩色的，每张图片的大小都是32\*32\*3，3代表R/G/B三个通道，每个像素点的颜色由R/G/B三个值决定，R/G/B的取值范围为0~255。仿照MNIST手写体数字识别，用TensorFlow 2.0框架实现卷积神经网络对CIFAR-10进行分类实验。

# -\*- coding: utf-8 -\*-

'''

@File    :   8.1.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import numpy as np

import pickle

import os

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.utils import to\_categorical

def load\_cifar10\_data(data\_dir):

    x\_train, y\_train = [], []

    x\_test, y\_test = [], []

    # 加载训练数据

    for i in range(1, 6):

        batch\_file = os.path.join(data\_dir, f'data\_batch\_{i}')

        print(f"Loading {batch\_file}")

        with open(batch\_file, 'rb') as file:

            batch = pickle.load(file, encoding='latin1')

            x\_train.append(batch['data'])

            y\_train += batch['labels']

    x\_train = np.concatenate(x\_train)

    x\_train = x\_train.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1).astype('float32') / 255.0

    y\_train = to\_categorical(y\_train, num\_classes=10)

    # 加载测试数据

    test\_batch\_file = os.path.join(data\_dir, 'test\_batch')

    print(f"Loading {test\_batch\_file}")

    with open(test\_batch\_file, 'rb') as file:

        batch = pickle.load(file, encoding='latin1')

        x\_test = batch['data']

        y\_test = batch['labels']

    x\_test = x\_test.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1).astype('float32') / 255.0

    y\_test = to\_categorical(y\_test, num\_classes=10)

    return (x\_train, y\_train), (x\_test, y\_test)

# 设置数据目录

data\_dir = r'D:\系统默认\文档\Tencent Files\2264162324\FileRecv\MobileFile\cifar-10-python\cifar-10-batches-py'

# 加载数据

try:

    (x\_train, y\_train), (x\_test, y\_test) = load\_cifar10\_data(data\_dir)

    print(f'Training data shape: {x\_train.shape}, Training labels shape: {y\_train.shape}')

    print(f'Testing data shape: {x\_test.shape}, Testing labels shape: {y\_test.shape}')

except FileNotFoundError as e:

    print(e)

# 构建卷积神经网络模型

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  # 第一层卷积层

model.add(layers.MaxPooling2D((2, 2)))  # 池化层

model.add(layers.Conv2D(64, (3, 3), activation='relu'))  # 第二层卷积层

model.add(layers.MaxPooling2D((2, 2)))  # 池化层

model.add(layers.Conv2D(64, (3, 3), activation='relu'))  # 第三层卷积层

model.add(layers.Flatten())  # 展平层

model.add(layers.Dense(64, activation='relu'))  # 全连接层

model.add(layers.Dense(10, activation='softmax'))  # 输出层，10类分类

# 编译模型

model.compile(optimizer='adam',

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# 训练模型

model.fit(x\_train, y\_train, epochs=10, batch\_size=64, validation\_split=0.1)

# 评估模型

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print(f'\nTest accuracy: {test\_acc:.4f}')

1. 试计算代码8-1所示例的卷积神经网络中各层需要学习的参数数量。

# -\*- coding: utf-8 -\*-

'''

@File    :   8.2.py

@Time    :   2024/11/24 14:00:00

@Author  :   210162401014 靳琛飞

'''

import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import datetime

# 设置随机种子

torch.manual\_seed(0)

# 下载MNIST数据集并进行预处理

transform = transforms.Compose([

    transforms.ToTensor(),

    transforms.Normalize((0.5,), (0.5,))

])

train\_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train\_loader = DataLoader(train\_dataset, batch\_size=200, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=200, shuffle=False)

# 定义CNN模型

class CNN(nn.Module):

    def \_\_init\_\_(self):

        super(CNN, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(in\_channels=1, out\_channels=32, kernel\_size=5)

        self.pool = nn.MaxPool2d(kernel\_size=2)

        self.dropout = nn.Dropout(0.2)

        self.fc1 = nn.Linear(32\*12\*12, 128)  # 输入特征数应根据卷积层输出调整

        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):

        x = self.pool(nn.functional.relu(self.conv1(x)))

        x = self.dropout(x)

        x = x.view(-1, 32\*12\*12)  # 将数据展平

        x = nn.functional.relu(self.fc1(x))

        x = nn.functional.softmax(self.fc2(x), dim=1)

        return x

model = CNN()

# 定义损失函数和优化器

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters())

# 计算每一层的参数数量

total\_params = sum(p.numel() for p in model.parameters())

print("总共需要学习的参数数量: ", total\_params)

print("每一层的参数数量:")

for i, (name, param) in enumerate(model.named\_parameters()):

    print(f"Layer {i}: {name} - Params: {param.numel()}")

startdate = datetime.datetime.now()  # 获取当前时间

# 训练模型

for epoch in range(2):

    model.train()

    for images, labels in train\_loader:

        optimizer.zero\_grad()

        outputs = model(images)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

enddate = datetime.datetime.now()

print("训练用时：" + str(enddate - startdate))

注：同学们请将你完成的作业，标明第X次作业、第Y章第Z题，按照序号放入同一个word文件，以“学号姓名”命名，提交给班长，班长收齐后，刻录一张光盘，在13周周末提交。