iris

April 29, 2023

1 Code for the implementation of task IRIS

1.0.1 Importing libraries

[ ]:
```
# For computation
import numpy as np
# For confusion matrices and plotting
from sklearn import metrics
import matplotlib.pyplot as plt
```

1.1 1.a)

Defining classes and loading function

[ ]:
```
class Dataset:
    def __init__(self, instances):
        data = {"training": {"targets": [], "features":[]}, "testing":␣
 ↪{"targets": [], "features":[]}}
        labelToTarget = {"Iris-setosa": [1, 0, 0], "Iris-versicolor": [0, 1,␣
 ↪0], "Iris-virginica": [0, 0, 1]}
        for instance in instances:
            match instance.set:
                case 'training':
                    data["training"]["targets"].append(labelToTarget[instance.
 ↪label])
                    data["training"]["features"].append(instance.features)
                case 'testing':
                    data["testing"]["targets"].append(labelToTarget[instance.
 ↪label])
                    data["testing"]["features"].append(instance.features)

        # convert to numpy array
        data["training"]["targets"] = np.array(data["training"]["targets"]).
 ↪astype(float)
        data["training"]["features"] = np.array(data["training"]["features"]).
 ↪astype(float)
```

```
        data["testing"]["targets"] = np.array(data["testing"]["targets"]).
↪astype(float)
        data["testing"]["features"]  = np.array(data["testing"]["features"]).
↪astype(float)

        self.data = data
        self.classes_names = ["Iris-setosa", "Iris-versicolor",␣
↪"Iris-virginica"]
        self.feature_names = ['Sepal length [cm]', 'Sepal width [cm]', 'Petal␣
↪length [cm]', 'Petal width [cm]']
        self.colors = ['red', 'green', 'blue']
        self.DESCR = "Iris plants dataset"
```

[ ]:
```
class Instance:
    def __init__(self, features, label, set):
        self.features = features
        self.label = label
        self.set = set
```

[ ]:
```
def loadDataSet(features_list: list):
    Instances = []
    path = "IRIS_TTT4275/iris.data"
    n_classes = 3
    n_training = 30
    n_testing  = 20
    with open(path) as file:
        for _ in range(n_classes):
            for _ in range(n_training):
                line = file.readline()
                line = line.split(',')
                features = []
                for i in features_list:
                    features.append(line[i])
                label = line[-1].strip("\n")
                training_instance = Instance(features=features, label=label,␣
↪set="training")
                Instances.append(training_instance)
            for _ in range(n_testing):
                line = file.readline()
                line = line.split(',')
                features = []
                for i in features_list:
                    features.append(line[i])
                label = line[-1].strip("\n")
                testing_instance = Instance(features=features, label=label,␣
↪set="testing")
```

```
                Instances.append(testing_instance)
    return Dataset(Instances)
```

loading the iris-dataset with all features

[ ]:
```
IRIS_Dataset = loadDataSet([0, 1, 2, 3])
```

1.1.1 b) Training a linear classifier

[ ]:
```
def sigmoid(x):                 # Activation function
    return 1/(1+np.exp(-x))
```

[ ]:
```
def TrainClassifier(dataset: Dataset, step_length: float, n_iterations, W_0: np.
 ↪ndarray):
    W = W_0
    MSEs = []
    n_classes = 3
    n_training = 30
    for _ in range(n_iterations):
        Gradient_MSE = np.zeros(W_0.shape)
        MSE = 0
        for index in range(n_classes * n_training):
            x_k = dataset.data["training"]["features"][index]
            x_k = np.append(x_k, 1)
            z_k = np.dot(W, x_k)
            g_k = sigmoid(z_k)
            t_k = dataset.data["training"]["targets"][index]
            MSE += 1/2 * np.dot((g_k - t_k).T,(g_k - t_k))
            Gradient_MSE += np.outer((g_k - t_k)*g_k*(np.ones((1,3))-g_k), x_k.
 ↪T)
        W = W - step_length * Gradient_MSE
        MSEs.append(MSE)
    print(f"MSE in first iteration {MSEs[0]}, and last iteration {MSEs[-1]}\n")
    print(f"Our W is \n{W}\n")
    return W, MSEs
```

For deciding the step-length, plotted in the report. Uncomment to see it

[ ]:
```
# MSEs_with_diff_step_lengths = []
# step_lengths = [0.00001, 0.001, 0.01, 0.09, 0.3, 0.6]
# n = 10000
# for alpha in step_lengths:
#     MSEs_with_diff_step_lengths.append(TrainClassifier(IRIS_Dataset, alpha,␣
 ↪n, np.zeros((3, 5)))[1])
# plt.figure(figsize=(10, 10))
# for index ,MSEs in enumerate(MSEs_with_diff_step_lengths):
#     plt.plot(MSEs, label=f' = {str(step_lengths[index])}')
```

```
# plt.legend(fontsize=12)
# plt.title("MSE for different step lengths", fontsize=20)
# plt.xlabel('Iterations', fontsize=15)
# plt.ylabel('Mean Square Error', fontsize=15)
# plt.savefig(f"svg_figures/MSEvsAlpha.svg")
# plt.show()
```

Hyperparameters for the best classifier See the one above

[ ]:
```
alpha = 0.01
n = 10000
```

[ ]:
```
W_0 = np.zeros((3, 5))
W, _ = TrainClassifier(IRIS_Dataset, alpha, n, W_0)
```

1.1.2 1c) training part, confusion matrix, error rate

[ ]:
```
def TestClassifier(dataset: Dataset, W: np.ndarray):
    n_classes = 3
    n_training = 30
    n_testing = 20
    ConfMatrices = {"training": np.zeros((n_classes, n_classes)), "testing": np.
↪zeros((n_classes, n_classes)), "fileName": dataset.DESCR}
    errors = {"training": 0, "testing": 0}
    for index in range(n_classes * n_testing):
        x_k = IRIS_Dataset.data["testing"]["features"][index]
        x_k = np.append(x_k, 1)
        t_k = IRIS_Dataset.data["testing"]["targets"][index]
        g_k = np.dot(W, x_k)
        ConfMatrices["testing"][np.argmax(t_k), np.argmax(g_k)] += 1
        if np.argmax(t_k) != np.argmax(g_k):
            errors["testing"] += 1

    for index in range(n_classes * n_training):
        x_k = IRIS_Dataset.data["training"]["features"][index]
        x_k = np.append(x_k, 1)
        t_k = IRIS_Dataset.data["training"]["targets"][index]
        g_k = np.dot(W, x_k)
        ConfMatrices["training"][np.argmax(t_k), np.argmax(g_k)] += 1
        if np.argmax(t_k) != np.argmax(g_k):
            errors["training"] += 1

    print(f"Error-rates: \nTraining: {errors['training']/
↪(n_training*n_classes)}, Testing: {errors['testing']/
↪(n_classes*n_testing)}\n")
    return ConfMatrices
```

[ ]:
```python
def PlotConfusionMatrix(matrix, fileName: str):
    plt.figure()

    fig, ax = plt.subplots(1, 2, figsize=(16,8))

    ax[0].set_title("Training set",fontsize=22)
    ax[1].set_title("Testing set", fontsize=22)
    metrics.ConfusionMatrixDisplay(confusion_matrix=matrix["training"],
                                   display_labels=IRIS_Dataset.classes_names,
                                   ).plot(ax=ax[0], cmap="cividis")
    metrics.ConfusionMatrixDisplay(confusion_matrix=matrix["testing"],
                                   display_labels=IRIS_Dataset.classes_names,
                                   ).plot(ax=ax[1], cmap="cividis")
    plt.tight_layout()
    plt.suptitle(f"{fileName}", fontsize=20, x=0.11)
    plt.savefig(f"svg_figures/{fileName}.svg")
    plt.show()
```

[ ]:
```python
ConfMatrix = TestClassifier(IRIS_Dataset, W)
PlotConfusionMatrix(ConfMatrix, "Iris_allFeaturesTrainFirst")
```

1.1.3 d) Switching ordering.

[ ]:
```python
# a little bit hard coded. Since we should only use it once
path = "IRIS_TTT4275/iris.data"
n_classes = 3
n_training = 30
n_testing  = 20

Instances = []

with open(path) as file:
    for _ in range(n_classes):
        for _ in range(n_testing):
            line = file.readline()
            line = line.split(',')
            features = line[0:-1]
            label = line[-1].strip("\n")
            testing_instance = Instance(features=features, label=label,␣
 ↪set="testing")
            Instances.append(testing_instance)

        for _ in range(n_training):
            line = file.readline()
            line = line.split(',')
            features = line[0:-1]
            label = line[-1].strip("\n")
```

```
            training_instance = Instance(features=features, label=label,␣
↪set="training")
            Instances.append(training_instance)
IRIS_Dataset = Dataset(Instances)
```

[ ]:
```
W_0 = np.zeros((3, 5))

W = W_0
MSEs = []
for i in range(n):
    Gradient_MSE = np.zeros(W_0.shape)
    MSE = 0
    for index in range(n_classes * n_training):
        x_k = IRIS_Dataset.data["training"]["features"][index]
        x_k = np.append(x_k, 1)
        z_k = np.dot(W, x_k)
        g_k = sigmoid(z_k)
        t_k = IRIS_Dataset.data["training"]["targets"][index]
        MSE += 1/2 * np.dot((g_k - t_k).T,(g_k - t_k))
        Gradient_MSE += np.outer((g_k - t_k)*g_k*(np.ones((1,3))-g_k), x_k.T)
    W = W - alpha * Gradient_MSE
    MSEs.append(MSE)
print(f"MSE in first iteration {MSEs[0]}, and last iteration {MSEs[-1]}\n")
print(f"Our W is {W}\n")
```

[ ]:
```
ConfMatrix = {"training": np.zeros((n_classes, n_classes)), "testing": np.
↪zeros((n_classes, n_classes)), "fileName": IRIS_Dataset.DESCR}
errors = {"training": 0, "testing": 0}
for index in range(n_classes * n_testing):
    x_k = IRIS_Dataset.data["testing"]["features"][index]
    x_k = np.append(x_k, 1)
    t_k = IRIS_Dataset.data["testing"]["targets"][index]
    g_k = np.dot(W, x_k)
    ConfMatrix["testing"][np.argmax(t_k), np.argmax(g_k)] += 1
    if np.argmax(t_k) != np.argmax(g_k):
        errors["testing"] += 1

for index in range(n_classes * n_training):
    x_k = IRIS_Dataset.data["training"]["features"][index]
    x_k = np.append(x_k, 1)
    t_k = IRIS_Dataset.data["training"]["targets"][index]
    g_k = np.dot(W, x_k)
    ConfMatrix["training"][np.argmax(t_k), np.argmax(g_k)] += 1
    if np.argmax(t_k) != np.argmax(g_k):
        errors["training"] += 1
print(f"Error-rates: \nTraining: {errors['training']/(n_training*n_classes)},␣
↪Testing: {errors['testing']/(n_classes*n_testing)}\n")
```

```
PlotConfusionMatrix(ConfMatrix, "Iris_allFeaturesTrainLast")
```

1.1.4 2a) Histograms

[ ]:
```
# loading data
n_examples = 50
paths = ["IRIS_TTT4275/class_1", "IRIS_TTT4275/class_2", "IRIS_TTT4275/class_3"]
features = {paths[0]: [[], [], [], []], paths[1]:[[], [], [], []], paths[2]:
 ↪[[], [], [], []]}
for i in range(len(paths)):
    with open(paths[i]) as file:
        for j in range(n_examples):
            line = file.readline().strip('\n').split(',')
            for k in range(len(line)):
                features[paths[i]][k].append(float(line[k]))
```

[ ]:
```
# displaying data
n_bars = 6
opacity = 0.4
plt.figure(figsize=(13,13))
plt.suptitle("Histograms of Iris dataset features", fontsize=28)
for index, feature in enumerate(IRIS_Dataset.feature_names):
    plt.subplot(2, 2, index+1)
    for class_nr in range(n_classes):
        plt.title(f"feature #{index+1}, {feature[:-5]}", fontsize=18)
        plt.hist(features[paths[class_nr]][index], label=IRIS_Dataset.
 ↪classes_names[class_nr], bins=n_bars, alpha=opacity, edgecolor='black',␣
 ↪color=IRIS_Dataset.colors[class_nr])
    plt.legend()
    plt.xlabel(feature, fontsize=18)
    plt.ylabel('count  [1]',fontsize=18)
    plt.grid()
plt.tight_layout()
plt.savefig("svg_figures/Iris_histograms.svg")
plt.show()
```

As Can be seen by the plots, feature 'Sepal width' is caotic and should be excluded. This is done below. Afterwards we scratch sepal length, the petal length

[ ]:
```
IRIS_Dataset = loadDataSet([0, 2, 3])
W_0 = np.zeros((3, 4))
W, _ = TrainClassifier(IRIS_Dataset, alpha, n, W_0)
ConfMatrix = TestClassifier(IRIS_Dataset, W)
PlotConfusionMatrix(ConfMatrix, "ThreeFeatures")
```

2b) with two features Petal width & length

```
[ ]: IRIS_Dataset = loadDataSet([2, 3])
W_0 = np.zeros((3, 3))
W, _ = TrainClassifier(IRIS_Dataset, alpha, n, W_0)
ConfMatrix = TestClassifier(IRIS_Dataset, W)
PlotConfusionMatrix(ConfMatrix, "TwoFeatures")
```

And with only one feature (Petal width):

```
[ ]: IRIS_Dataset = loadDataSet([3])
W_0 = np.zeros((3, 2))
W, _ = TrainClassifier(IRIS_Dataset, alpha, n, W_0)
ConfMatrix = TestClassifier(IRIS_Dataset, W)
PlotConfusionMatrix(ConfMatrix, "OneFeature")
```

1.1.5 2d) see report part.

Making of scatter plot, also commented out

```
[ ]: # n_classes = 3
# plt.figure(figsize=(13,13))
# for index in range(n_classes):
#     plt.scatter(features[paths[index]][2], features[paths[index]][3],␣
↪label=IRIS_Dataset.classes_names[index], color=IRIS_Dataset.colors[index],␣
↪s=200)
# plt.legend(fontsize=20)
# plt.grid()
# plt.xlabel("Feature #3, Petal length [cm]", fontsize=20)
# plt.ylabel("Feature #4 Petal width [cm]", fontsize=20)
# plt.title("Scatter plot of Iris dataset", fontsize=30)
# plt.savefig("svg_figures/Iris_scatter.svg")
# plt.show()
```