2.Lab: Introduction to regression and gradient descent with Python

1. Learning objectives

1. Understand the basics of supervised learning.
2. Learn to work with simple datasets.
3. Implement a basic regression model for prediction.
4. Implement a basic regression model for classification.
5. Experiment with different learning rates.
6. Evaluate model performance using appropriate metrics.

2. Linear regression

We will use the available *California housing dataset*. Create a Python project for this assignment.

1. Install pandas, scikit-learn, numpy and matplotlib (you can use pip or a python script).
2. Import the libraries to your project. From sklearn, we will need the dataset and preprocessing libraries. From matplotlib we will need pyplot.

```python
# Importing pandas, scikit-learn, numpy and matplotlib
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn import preprocessing
import matplotlib.pyplot as plt
```

3. From sci-kit learn, import and load the [California Housing dataset](#).

```python

# Load the California Housing dataset

california_housing = datasets.fetch_california_housing()

```

```
# Display basic information about the dataset
print(f"Features: {california_housing.feature_names}")
print(f"Target: {california_housing.target_names}")
print(f"Shape of data: {california_housing.data.shape}")
print(f"Shape of target: {california_housing.target.shape}")
```

4. Check the documentation for the dataset.
   What does the dataset contain?
   What does each datapoint represent?
   How many datapoints does the dataset contain? How many features does the dataset contain?
5. Visualise and study the dataset:
   - Plot the feature distributions and relationships between features.
   - Plot housing prices against a few selected features.

   Include the plots in your report. Is it reasonable to assume that the relationship between housing prices and the other features is linear?
6. Implement a linear regression with gradient descent model to predict housing prices based on the features of the dataset.

```Python
# Prepare the data
X = california_housing.data
y = california_housing.target

# Normalize the features
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X)

# Add a column of ones to X for the intercept term
X = np.c_[np.ones(X.shape[0]), X]
# X.shape[0] is the number of rows (i.e. number of datapoints)

# Initialize parameters
theta = np.zeros(X.shape[1])

# X.shape[1] is the number of columns (i.e. number of features)
#np.zeros(X.shape[1]) will create an array [0, 0,..., 0] as long as the
number of columns
#we initialise all weights as 0

# Hyperparameters
learning_rate = 0.01
```

```
num_iterations = 1000

# Gradient Descent Function
def gradient_descent(X, y, theta, learning_rate, num_iterations):
    m = len(y) #m: The number of training examples (length of y, or number
of rows in X)
    cost_history = [] #will store the error at each iteration

    for i in range(num_iterations):
        predictions = X.dot(theta) #dot product of X and theta, gives a
prediction for each datapoint in X
        errors = predictions - y
        gradients = (2/m) * X.T.dot(errors) #gradient: dot product of the
transpose of X and of the error vector (you can check manually why this
corresponds to the sum formula seen in the slides)
        theta -= learning_rate * gradients #update weights
        cost = (1/m) * np.sum(errors ** 2) #cost: squared mean errors
        cost_history.append(cost) #store cost

        if i % 100 == 0:
            print(f"Iteration {i}: Cost {cost}") #print cost every 100
iterations

    return theta, cost_history

# Run gradient descent
theta, cost_history = gradient_descent(X, y, theta, learning_rate,
num_iterations)

# Plotting the cost function history
plt.plot(range(num_iterations), cost_history, 'b-')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Cost Function History')
plt.show()

# Make predictions
predictions = X.dot(theta)

# Display the first 5 predictions
print("First 5 predictions:", predictions[:5])
print("First 5 actual values:", y[:5])
```

7. Experiment with Learning Rates:
    - Test at least 5 different learning rates (e.g., 0.001, 0.01, 0.1, 0.5, 1).
    - Track the cost function at each iteration.

- Plot the cost function value versus the number of iterations for each learning rate.
- Compare the convergence speed and final cost for each learning rate.

What is the impact of different learning rates on model performance and convergence?

8. Choose one model to evaluate: Calculate the mean square error and R squared error.

What are the errors? Why are they different?

4. Logistic regression

1. From sci-kit learn, import and load the Iris dataset.
2. Check the documentation for the dataset.
   What does the dataset contain?
   What does each datapoint represent?
   How many datapoints does the dataset contain? How many features does the dataset contain?
3. Visualise and study the dataset. You can try the seaborn library (if so, don't forget to install and load it).
   - You can try scatterplots, heatmaps, boxplots…
   - To do so using the pandas library, you are going to need to transform the dataset into a dataframe

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
 #Load iris dataset and store as a dataframe
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

#rename target variable
iris_df['species'] = pd.Categorical.from_codes(iris.target,
iris.target_names)

# Display basic statistics of the dataframe
print(iris_df.describe())

# Pairplot to visualize relationships between features
sns.pairplot(iris_df, hue='species')
plt.suptitle('Pairplot of Iris Dataset', y=1.02)
plt.show()
```

```
# Boxplot to visualize the distribution of each feature
plt.figure(figsize=(10, 6))
sns.boxplot(data=iris_df)
plt.title('Boxplot of Iris Features')
plt.xticks(rotation=90)
plt.show()

# Violin plot to visualize the distribution and density of each feature for
each species
plt.figure(figsize=(10, 6))
for feature in iris.feature_names:
    plt.figure()
    sns.violinplot(x='species', y=feature, data=iris_df)
    plt.title(f'Violin Plot of {feature}')
    plt.show()

# Scatter plot matrix with KDE (Kernel Density Estimate)
sns.pairplot(iris_df, hue='species', kind='kde', diag_kind='kde',
markers=['o', 's', 'D'])
plt.suptitle('Scatter Plot Matrix with KDE of Iris Dataset', y=1.02)
plt.show()
```

What does the visualisation tell you?

4. Train a Logistic Regression Model to classify the irises.

Python
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# Split the data into training and testing sets
X = iris_df.drop('species', axis=1)
y = iris_df['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

# Train a Logistic Regression model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Look into the functions you used. How does logistic regression work?

5. Evaluate the model using a confusion matrix.

```python
# Print confusion matrix
print('Confusion Matrix:')
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Look into the functions you used. What is a confusion matrix?