Programming Machine Learning Lab

Exercise 11

General Instructions:

1. You need to submit the PDF as well as the filled notebook file.
2. Name your submissions by prefixing your matriculation number to the filename. Example, if your MR is 12345 then rename the files as **"12345_Exercise_11.xxx"**
3. Complete all your tasks and then do a clean run before generating the final PDF. (*Clear All Ouputs* and *Run All* commands in Jupyter notebook)

Exercise Specific instructions::

1. You are allowed to use only NumPy and Pandas (unless stated otherwise). You can use any library for visualizations.

Part 1

In this part, you will be using the credit card fraud detection dataset from https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud to train and test a Support Vector Machine (SVM) classifier. Your task is to:

1. Download the data and split the dataset into training and testing sets (80-20 split) in a stratified manner to take care of the class imbalance. You need to code the stratified splitting function from scratch. *sklearn is not allowed for this part*
2. Implement the basic Pegasos Algorithm from the paper https://home.ttic.edu/~nati/Publications/PegasosMPB.pdf. This is in page 5, Fig 1.
3. Implement the mini-batch Pegasos algorithm from the paper https://home.ttic.edu/~nati/Publications/PegasosMPB.pdf. Do not forget the projection step. This is in page 6, Fig 2.
4. Implement the dual coordinate descent method for SVM's from the paper https://icml.cc/Conferences/2008/papers/166.pdf. This is Algorithm 1 in the paper.
5. Report a final accuracy on the test set for all 3 approches.

In [ ]:
```
### Write your code here
import pandas as pd
import numpy as np
```

In [ ]:
```
df = pd.read_csv('creditcard.csv', header=0, index_col=None)
df.head()
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.27783
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.63867
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.77167
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.00527
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.79827

5 rows × 31 columns

In [ ]:
```
def stratified_split(df, test_size=0.2, random_state=None):
    # Shuffle the dataframe
    df_shuffled = df.sample(frac=1, random_state=random_state)

    # Separate the data into fraud (class 1) and non-fraud (class 0)
    fraud_data = df_shuffled[df_shuffled['Class'] == 1]
    non_fraud_data = df_shuffled[df_shuffled['Class'] == 0]

    # Calculate the number of samples for each class in the test set
    test_size_fraud = int(test_size * len(fraud_data))
    test_size_non_fraud = int(test_size * len(non_fraud_data))

    # Split the data for each class into training and testing
    fraud_train = fraud_data.iloc[:-test_size_fraud, :]
    fraud_test = fraud_data.iloc[-test_size_fraud:, :]

    non_fraud_train = non_fraud_data.iloc[:-test_size_non_fraud, :]
    non_fraud_test = non_fraud_data.iloc[-test_size_non_fraud:, :]
```

```python
    # Concatenate the training and testing sets for both classes
    train_set = pd.concat([fraud_train, non_fraud_train])
    test_set = pd.concat([fraud_test, non_fraud_test])

    # Shuffle the training and testing sets again
    train_set = train_set.sample(frac=1, random_state=random_state).reset_index(drop=True)
    test_set = test_set.sample(frac=1, random_state=random_state).reset_index(drop=True)

    return train_set, test_set

# Usage
train_set, test_set = stratified_split(df, test_size=0.2, random_state=42)
```

In [ ]:

```python
def pegasos_svm(X, y, lambda_reg, T):
    m, n = X.shape
    w = np.zeros(n)
    for t in range(1, T + 1):
        i = np.random.randint(0, m)  # Randomly pick a data point
        eta = 1 / (lambda_reg * t)

        if y[i] * np.dot(X[i], w) < 1:
            w = (1 - eta * lambda_reg) * w + eta * y[i] * X[i]
        else:
            w = (1 - eta * lambda_reg) * w

    return w

def pegasos_svm_predict(X, weights):
    # Predict using the learned weights
    return np.sign(np.dot(X, weights))

X = train_set.drop('Class', axis=1).values
y = train_set['Class'].values
y = np.where(y == 0, -1, 1)  # Convert 0 labels to -1 for SVM
X_test = test_set.drop('Class', axis=1).values
y_test = test_set['Class'].values
y_test = np.where(y_test == 0, -1, 1)  # Convert 0 labels to -1 for SVM

# Set hyperparameters
lambda_reg = 0.01
T = 1000  # Number of iterations
```

```python
# Train SVM using Pegasos
weights = pegasos_svm(X, y, lambda_reg, T)
predictions = pegasos_svm_predict(X_test, weights)

# Calculate accuracy
accuracy = np.mean(predictions == y_test)
print("Accuracy on the test set:", accuracy)
```

```
Accuracy on the test set: 0.9982795245869981
```

In [ ]:

```python
def mini_batch_pegasos_svm(X, y, lambda_reg, T, batch_size):
    m, n = X.shape
    w = np.zeros(n)
    t = 0

    for _ in range(T):
        indices = np.random.choice(m, batch_size, replace=False)  # Randomly select a mini-batch
        X_batch, y_batch = X[indices], y[indices]
        t += 1

        eta = 1 / (lambda_reg * t)

        # Calculate the hinge loss gradient
        misclassified_indices = np.where(y_batch * np.dot(X_batch, w) < 1)[0]

        # Perform the update with the mini-batch gradient
        w = (1 - eta * lambda_reg) * w + (eta / batch_size) * \
                np.sum(y_batch[misclassified_indices, np.newaxis] * X_batch[misclassified_indices], axis=0)

        # Projection step
        w_norm = np.linalg.norm(w)
        if w_norm > 1 / np.sqrt(lambda_reg):
            w = w / (np.sqrt(lambda_reg) * w_norm)

    return w

batch_size = 64  # Size of each mini-batch

# Train SVM using Mini-Batch Pegasos
weights_mini_batch = mini_batch_pegasos_svm(X, y, lambda_reg, T, batch_size)
```

```python
def mini_batch_pegasos_svm_predict(X, weights):
    # Predict using the learned weights
    return np.sign(np.dot(X, weights))

predictions = mini_batch_pegasos_svm_predict(X_test, weights_mini_batch)

# Calculate accuracy
accuracy = np.mean(predictions == y_test)
print("Accuracy on the test set:", accuracy)
```

```
Accuracy on the test set: 0.9982795245869981
```

In [ ]:

```python
def dual_coordinate_descent_svm(X, y, C, epochs):
    m, n = X.shape
    alpha = np.zeros(m)
    w = np.zeros(n)

    for epoch in range(1, epochs + 1):
        for i in range(m):
            xi, yi = X[i], y[i]

            # Compute wTxi
            wtxi = np.dot(w, xi)

            # Compute G
            G = yi * wtxi - 1 + alpha[i]

            # Compute PG
            if alpha[i] == 0:
                PG = min(G, 0)
            elif alpha[i] == C:
                PG = max(G, 0)
            else:
                PG = G

            # Update alpha[i]
            if not np.abs(PG) == 0:
                alpha_prime_i = alpha[i]
                alpha[i] = min(max(alpha[i] - G / (np.linalg.norm(xi) ** 2), 0), C)

                # Update w
                w += (alpha[i] - alpha_prime_i) * yi * xi
```

```python
    return w

C = 1.0
epochs = 10

# Train SVM using Dual Coordinate Descent
weights_dual = dual_coordinate_descent_svm(X, y, C, epochs)

def dual_coordinate_descent_svm_predict(X, weights):
    # Predict using the learned weights
    return np.sign(np.dot(X, weights))

predictions = dual_coordinate_descent_svm_predict(X_test, weights_dual)

# Calculate accuracy
accuracy = np.mean(predictions == y_test)
print("Accuracy on the test set:", accuracy)
```

```
Accuracy on the test set: 0.9982619687154369
```

In [ ]: