PrivacyProtectionFL_CodingAssignment

June 25, 2024

1 Coding Assignment - "Privacy-Protection in FL"

1.1 1. Preparation

1.1.1 1.1 Libraries

[ ]:
```
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

1.1.2 1.2 Helper functions

[ ]:
```
# The function below extracts features and labels
# from each row of a dataframe.
# Each row is expected to hold an FMI weather measurement
# with columns "Latitude", "Longitude", "temp", "Timestamp".
# Returns numpy arrays X, y.
def ExtractFeatureMatrixLabelVector(data):
    n_features = 7
    n_datapoints = len(data)

    # We build the feature matrix X (each of its rows hold the features of data␣
↪points)
    # and the label vector y (whose entries hold the labels of data points).
    X = np.zeros((n_datapoints, n_features))
    y = np.zeros((n_datapoints, 1))

    # Iterate over all rows in dataframe and create the corresponding feature␣
↪vector and label.
    for i in range(n_datapoints):
        # Latitude of FMI station, normalized by 100.
        lat = float(data['Latitude'].iloc[i])/100
        # Longitude of FMI station, normalized by 100.
        lon = float(data['Longitude'].iloc[i])/100
```

```
        # Temperature value of the data point.
        tmp = data['temp'].iloc[i]
        # Read the date and time of the temperature measurement.
        date_object = datetime.strptime(data['Timestamp'].iloc[i], '%Y-%m-%d %H:
 ↪%M:%S')
        # Extract year, month, day, hour, and minute. Normalize these values
        # to ensure that the features are in range [0,1].
        year = float(date_object.year)/2025
        month = float(date_object.month)/13
        day = float(date_object.day)/32
        hour = float(date_object.hour)/25
        minute = float(date_object.minute)/61

        # Store the data point's features and a label.
        X[i,:] = [lat, lon, year, month, day, hour, minute]
        y[i,:] = tmp

    return X, y
```

1.2 2. Data

1.2.1 2.1 Dataset

[ ]:
```
# Import the weather measurements.
data = pd.read_csv('Assignment_MLBasicsData.csv')

# We consider each temperature measurement (=a row in dataframe) as a
# separate data point.

# Define the numbers of data points, the unique stations, and features.
n_stations = len(data.name.unique())
n_datapoints = len(data)
n_features = 7
```

1.2.2 2.2 Features and labels

[ ]:
```
# Extract features and labels from the FMI data.
X, y = ExtractFeatureMatrixLabelVector(data)

print(f"The feature matrix contains {np.shape(X)[0]} entries of {np.
 ↪shape(X)[1]} features each.")
print(f"The label vector contains {np.shape(y)[0]} measurements.")
```

1.3 3. Model

1.3.1 3.1 Where are you? - predict the latitude and longitude

[ ]:
```
###################TODO###################
# TODO: 1. Define features as year, month, day, hour,
#          minute, and temperature measurement.
#          The resulting feature matrix has the shape (19768, 6).
#       2. Define labels as latitude and longitude.
#          The resulting label matrix has the shape (19768, 2).
#       3. Split this data into training and validation sets.
#       4. Fit the Linear Regression to the training data.
#       5. Calculate training and validation errors.
#
# NOTE: 1. Split the data with train_test_split().
#          Use 0.2 for the test_size and 4740 for the random_state.
#       2. Use LinearRegression() sklearn class for the model fitting.
#          Intercept fitting is necessary: fit_intercept=True.
#       3. Use mean_squared_error() sklearn class for calculating the errors.

raise NotImplementedError

# Choose features and labels for the current task.
# X_time_temp =
# y_location =

# Split into training and validation sets.
# X_train, X_val, y_train, y_val =

# Train a Linear Regression.
# reg_original_data =
print("The model has been trained on the original data.")

# Calculate training and validation errors.
# train_error =
# val_error =
print(f"Training error:   {train_error}")
print(f"Validation error: {val_error}")
```

1.3.2 3.2 Ensuring Privacy with Pre-Processing

[ ]:
```
# Define the random seeds for testing.
seeds = [1, 4740, 5]
for seed in seeds:
    print(f"The random seed: {seed}")
    np.random.seed(seed)

    ###################TODO###################
```

```
    # TODO: 1. Add noise to the features and labels
    #          defined in the section 3.1.
    #       2. Split this data into training and validation sets.
    #       3. Fit the Linear Regression to the training noisy data.
    #       4. Calculate training and validation errors between the
    #          labels without noise and the labels predicted by the model,
    #          trained on the noisy data.
    #
    # NOTE: 1. The noise should follow the standard normal distribution,
    #          i.e., zero mean and unit variance.
    #       2. Split the data with train_test_split().
    #          Use 0.2 for the test_size and 48433 for the random_state.
    #       3. Use LinearRegression() sklearn class for the model fitting.
    #          Intercept fitting is necessary: fit_intercept=True.
    #       4. Use mean_squared_error() sklearn class for calculating the␣
↪errors.

    raise NotImplementedError

    # Add noise to the data used in the previous section.
    # X_time_temp_with_noise =
    # y_location_with_noise =

    # Split into training and validation sets.
    # Use different random state to train the model
    # on a different subset of datapoints.
    # X_train_with_noise, _, y_train_with_noise, _ =

    # Train a Linear Regression.
    # reg_perturbed_data =
    print("The model has been trained on the perturbed data.")

    # Calculate training and validation errors.
    # train_error =
    # val_error =
    print(f"Training error:   {train_error}")
    print(f"Validation error: {val_error}\n")
```

1.3.3 3.3 Ensuring Privacy with Post-Processing

[ ]:
```
# Define the random seeds for testing.
seeds = [1, 4740, 5]
for seed in seeds:
    print(f"The random seed: {seed}")
    np.random.seed(seed)

    ##################TODO###################
```

```
    # TODO: 1. Add noise to the model, trained on
    #          the data without noise.
    #       2. Predict the training and validation labels
    #          with the noisy model from the features without noise.
    #       3. Calculate training and validation errors between the
    #          labels without noise and the predicted labels.
    #
    # NOTE: 1. You can re-use the trained model from
    #          the section 3.1. Add the noise to its
    #          itercept term. The noise should follow
    #          the standard normal distribution,
    #          i.e., zero mean and unit variance.
    #       2. Use mean_squared_error() sklearn class for calculating the␣
↪errors.

    raise NotImplementedError

    # Make predictions.
    # y_train_pred =
     #y_val_pred =
    print("The model has been perturbed.")

    # Calculate training and validation errors.
    # train_error =
    # val_error =
    print(f"Training error:   {train_error}")
    print(f"Validation error: {val_error}\n")
```

1.3.4 3.4 Private Feature Learning

3.4.1 Privacy preserving features

[ ]:
```
# In this task we use location and time as features
# and temperature measurement as a label
# (i.e., the feature matrix and the label vector created in the section 2.2).

##################TODO###################
# TODO: 1. Centerlize the features.
#       2. Extract the private attributes (centred normalized latitude).
#       3. Compute the approximate cross-covariance vector.
#       4. Compute the linear feature map.
#       5. Use the computed map to get the privacy preserving features.
#
# NOTE: 1. See Lecture Notes 9.5.4 for formulas.

raise NotImplementedError

# Remove the sample means from each feature.
```

```
# X_centred =

# Extract the private attribute (centred normalized latitude)
# for each data point.
# s_centred =

# The approximate cross-covariance vector.
# c_hat =

# Linear feature map.
# F =

# Compute the privacy preserving features.
# Z =

# Sanity checks (must be all True).
print(X_centred.shape == (19768, 7))
print(s_centred.shape == (19768,))
print(c_hat.shape == (7, 1))
print(F.shape == (7, 7))
print(Z.shape == (19768, 7))
```

3.4.2 Private attribute prediction

[ ]:
```
###################TODO###################
# TODO: 1. Split the privacy preserving features
#          and centred private attribute
#          to the training and validation sets.
#       2. Fit the Linear Regression to the training data.
#       3. Calculate training and validation errors.
#
# NOTE: 1. Split the data with train_test_split().
#          Use 0.2 for the test_size and 4740 for the random_state.
#       2. Use LinearRegression() sklearn class for the model fitting.
#          Intercept fitting is necessary: fit_intercept=True.
#       3. Use mean_squared_error() sklearn class for calculating the errors.

raise NotImplementedError

# Split into training and validation sets.
# Z_train, Z_val, s_centred_train, s_centred_val =

# To measure the usefulness of the new features Z,
# we use them to train a predictor for the private attribute
# and hope that the resulting validation error is large.
# reg =
```

```
# train_error =
# val_error =

print("Train/Val errors obtained when using privacy-preserving features Z")
print("Training Error:", train_error / np.var(s_centred))
print("Validation Error:", val_error / np.var(s_centred))
print("\n*************************************************************\n")

##################
# lets redo the above training/validation using the original features X instead␣
↪of the Z
# and compute resulting training/validation errors when using the original␣
↪features X (instead of Z).
# We expect that the resulting validation error should be much smaller since X␣
↪leaks more information
# about the private attribute, compared to Z.
####################

###################TODO###################
# TODO: 1. Repeat the calculations with the original features.

raise NotImplementedError

# Split into training and validation sets.
# X_train, X_val, s_centred_train, s_centred_val =

# reg =

# train_error =
# val_error =

print("Train/Val errors obtained when using original features X")
print("Training Error:", train_error / np.var(s_centred))
print("Validation Error:", val_error / np.var(s_centred))
```

3.4.3 Labels prediction The code snippet below measures how useful the new (privacy preserving) features Z are for the ultimate goal, i.e., to predict the temperature of a weather recording.

[ ]:
```
###################TODO###################
# TODO: 1. Split the privacy preserving features
#          and the original labels
#          to the training and validation sets.
#       2. Fit the Linear Regression to the training data.
#       3. Calculate training and validation errors.
#
# NOTE: 1. Split the data with train_test_split().
```

```
#          Use 0.2 for the test_size and 4740 for the random_state.
#       2. Use LinearRegression() sklearn class for the model fitting.
#          Intercept fitting is necessary: fit_intercept=True.
#       3. Use mean_squared_error() sklearn class for calculating the errors.

raise NotImplementedError

# Split into training and validation sets.
# Z_train, Z_val, y_train, y_val =

# reg =

# train_error =
# val_error =

print("Train/Val errors obtained when using new features Z to predict tempature␣
 ↪y")
print("Training Error:", train_error)
print("Validation Error:", val_error)
print("\n************************************************************************\n")


###################TODO###################
# TODO: 1. Repeat the calculations with the original features.

raise NotImplementedError

# Split into training and validation sets.
# X_train, X_val, y_train, y_val =

# reg =

# train_error =
# val_error =

print("Train/Val errors obtained when using original (privacy-leaking) features␣
 ↪X to predict tempature y")
print("Training Error:", train_error)
print("Validation Error:", val_error)
```

[ ]:

```
```

PrivacyProtectionFL_RefSol

June 25, 2024

1 Reference Solution for Coding Assignment “Privacy-Protection in FL”

1.1 1. Preparation

1.1.1 1.1 Libraries

[1]:
```
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

1.1.2 1.2 Helper functions

[2]:
```
# The function below extracts features and labels
# from each row of a dataframe.
# Each row is expected to hold an FMI weather measurement
# with columns "Latitude", "Longitude", "temp", "Timestamp".
# Returns numpy arrays X, y.
def ExtractFeatureMatrixLabelVector(data):
    n_features = 7
    n_datapoints = len(data)

    # We build the feature matrix X (each of its rows hold the features of data␣
 ↪points)
    # and the label vector y (whose entries hold the labels of data points).
    X = np.zeros((n_datapoints, n_features))
    y = np.zeros((n_datapoints, 1))

    # Iterate over all rows in dataframe and create the corresponding feature␣
 ↪vector and label.
    for i in range(n_datapoints):
        # Latitude of FMI station, normalized by 100.
        lat = float(data['Latitude'].iloc[i])/100
        # Longitude of FMI station, normalized by 100.
```

```
        lon = float(data['Longitude'].iloc[i])/100
        # Temperature value of the data point.
        tmp = data['temp'].iloc[i]
        # Read the date and time of the temperature measurement.
        date_object = datetime.strptime(data['Timestamp'].iloc[i], '%Y-%m-%d %H:
 ↪%M:%S')
        # Extract year, month, day, hour, and minute. Normalize these values
        # to ensure that the features are in range [0,1].
        year = float(date_object.year)/2025
        month = float(date_object.month)/13
        day = float(date_object.day)/32
        hour = float(date_object.hour)/25
        minute = float(date_object.minute)/61

        # Store the data point's features and a label.
        X[i,:] = [lat, lon, year, month, day, hour, minute]
        y[i,:] = tmp

    return X, y
```

1.2 2. Data

1.2.1 2.1 Dataset

[3]:
```
# Import the weather measurements.
data = pd.read_csv('Assignment_MLBasicsData.csv')

# We consider each temperature measurement (=a row in dataframe) as a
# separate data point.

# Define the numbers of data points, the unique stations, and features.
n_stations = len(data.name.unique())
n_datapoints = len(data)
n_features = 7
```

1.2.2 2.2 Features and labels

[4]:
```
# Extract features and labels from the FMI data.
X, y = ExtractFeatureMatrixLabelVector(data)

print(f"The feature matrix contains {np.shape(X)[0]} entries of {np.
 ↪shape(X)[1]} features each.")
print(f"The label vector contains {np.shape(y)[0]} measurements.")
```

```
The feature matrix contains 19768 entries of 7 features each.
The label vector contains 19768 measurements.
```

1.3 3. Model

1.3.1 3.1 Where are you? - predict the latitude and longitude

[5]:
```python
# Choose features and labels for the current task.
X_time_temp = np.concatenate((X[:, 2:], y), axis=1)
y_location = X[:, :2]

# Split into training and validation sets.
X_train, X_val, y_train, y_val = train_test_split(X_time_temp,
                                                  y_location,
                                                  test_size=0.2,
                                                  random_state=4740)

# Train a Linear Regression.
reg_original_data = LinearRegression()
reg_original_data.fit(X_train, y_train)
print("The model has been trained on the original data.")

# Calculate training and validation errors.
train_error = mean_squared_error(y_train, reg_original_data.predict(X_train))
val_error = mean_squared_error(y_val, reg_original_data.predict(X_val))

print(f"Training error:   {train_error}")
print(f"Validation error: {val_error}")
```

```
The model has been trained on the original data.
Training error:   0.0005274675884900359
Validation error: 0.000535977207047403
```

1.3.2 3.2 Ensuring Privacy with Pre-Processing

[6]:
```python
seeds = [1, 4740, 5]
for seed in seeds:
    print(f"The random seed: {seed}")
    np.random.seed(seed)

    def add_gaussian_noise(data):
        std = 1
        mean = 0
        noise = np.random.normal(mean, std, data.shape)

        return data + noise

    # Add noise to the data used in the previous section.
    X_time_temp_with_noise = add_gaussian_noise(X_time_temp)
    y_location_with_noise = add_gaussian_noise(y_location)
```

```
    # Split into training and validation sets.
    # Use different random state to train the model
    # on a different subset of datapoints.
    X_train_with_noise, _, y_train_with_noise, _ = 
↪train_test_split(X_time_temp_with_noise,
                                                                                    
↪y_location_with_noise,
                                                                                     test_size=0.
↪2,
                                                                                    
↪random_state=48433)

    # Train a Linear Regression.
    reg_perturbed_data = LinearRegression()
    reg_perturbed_data.fit(X_train_with_noise, y_train_with_noise)
    print("The model has been trained on the perturbed data.")

    # Calculate training and validation errors.
    train_error = mean_squared_error(y_train, reg_perturbed_data.
↪predict(X_train))
    val_error = mean_squared_error(y_val, reg_perturbed_data.predict(X_val))

    print(f"Training error:   {train_error}")
    print(f"Validation error: {val_error}\n")
```

```
The random seed: 1
The model has been trained on the perturbed data.
Training error:   0.0007356887830373 59
Validation error: 0.0007432764642231133

The random seed: 4740
The model has been trained on the perturbed data.
Training error:   0.0011861485388892424
Validation error: 0.0011865235864328295

The random seed: 5
The model has been trained on the perturbed data.
Training error:   0.0006715936091527291
Validation error: 0.0006768943924952629
```

1.3.3 3.3 Ensuring Privacy with Post-Processing

[7]:
```
seeds = [1, 4740, 5]
for seed in seeds:
    print(f"The random seed: {seed}")
    np.random.seed(seed)
```

```python
def predict_with_noisy_hypothesis(train_features, model):
        # Add normal noise to the intercept.
        intercept = np.tile(model.intercept_, (train_features.shape[0], 1))
        intercept_with_noise = intercept + np.random.normal(0, 1, 2)
        prediction = train_features @ model.coef_.T + intercept_with_noise
        return prediction

# Make predictions.
y_train_pred = predict_with_noisy_hypothesis(X_train, reg_original_data)
y_val_pred = predict_with_noisy_hypothesis(X_val, reg_original_data)
print("The model has been perturbed.")

# Calculate training and validation errors.
train_error = mean_squared_error(y_train, y_train_pred)
val_error = mean_squared_error(y_val, y_val_pred)

print(f"Training error:   {train_error}")
print(f"Validation error: {val_error}\n")
```

```
The random seed: 1
The model has been perturbed.
Training error:   1.5068989600096512
Validation error: 0.7151264111083732

The random seed: 4740
The model has been perturbed.
Training error:   0.4077012595159399
Validation error: 0.008044551353045168

The random seed: 5
The model has been perturbed.
Training error:   0.15260571781756077
Validation error: 2.986456779574424
```

1.3.4 3.4 Private Feature Learning

3.4.1 Privacy preserving features

[8]:
```python
# Remove the sample means from each feature.
X_centred = X - np.mean(X, axis=0)

# Extract the private attribute (centred normalized latitude)
# for each data point.
s_centred = X_centred[:, 0]

# The approximate cross-covariance vector.
```

```python
c_hat = (np.dot(X_centred.T, s_centred) / n_datapoints).reshape(n_features,1)

# Linear feature map.
F = np.identity(n_features) - np.dot(c_hat, c_hat.T) / np.linalg.norm(c_hat) **␣
↪2

# Compute the privacy preserving features.
Z = np.dot(F, X_centred.T).T

# Sanity checks (must be all True).
print(X_centred.shape == (19768, 7))
print(s_centred.shape == (19768,))
print(c_hat.shape == (7, 1))
print(F.shape == (7, 7))
print(Z.shape == (19768, 7))
```

```
True
True
True
True
True
```

3.4.2 Private attribute prediction

[9]:
```python
# Split into training and validation sets.
Z_train, Z_val, s_centred_train, s_centred_val = train_test_split(Z, s_centred,␣
↪test_size=0.2, random_state=4740)

# To measure the usefulness of the new features Z,
# we use them to train a predictor for the private attribute
# and hope that the resulting validation error is large.

reg = LinearRegression()
reg.fit(Z_train, s_centred_train)

train_error = mean_squared_error(s_centred_train, reg.predict(Z_train))
val_error = mean_squared_error(s_centred_val, reg.predict(Z_val))

print("Train/Val errors obtained when using privacy-preserving features Z")
print("Training Error:", train_error / np.var(s_centred))
print("Validation Error:", val_error / np.var(s_centred))
print("\n**************************************************************\n")

##################
# lets redo the above training/validation using the original features X instead␣
↪of the Z
```

```
# and compute resulting training/validation errors when using the original␣
↪features X (instead of Z).
# We expect that the resulting validation error should be much smaller since X␣
↪leaks more information
# about the private attribute, compared to Z.
####################

# Split into training and validation sets.
X_train, X_val, s_centred_train, s_centred_val = train_test_split(X, s_centred,␣
↪test_size=0.2, random_state=4740)

reg = LinearRegression()
reg.fit(X_train, s_centred_train)

train_error = mean_squared_error(s_centred_train, reg.predict(X_train))
val_error = mean_squared_error(s_centred_val, reg.predict(X_val))

print("Train/Val errors obtained when using original features X")
print("Training Error:", train_error / np.var(s_centred))
print("Validation Error:", val_error / np.var(s_centred))
```

```
Train/Val errors obtained when using privacy-preserving features Z
Training Error: 1.003611902314061
Validation Error: 0.9857399699602243

*******************************************************************

Train/Val errors obtained when using original features X
Training Error: 3.719283221546538e-28
Validation Error: 3.665935665838089e-28
```

3.4.3 Labels prediction The code snippet below measures how useful the new (privacy preserving) features Z are for the ultimate goal, i.e., to predict the temperature of a weather recording.

[10]:
```
# Split into training and validation sets.
Z_train, Z_val, y_train, y_val = train_test_split(Z, y, test_size=0.2,␣
↪random_state=4740)

reg = LinearRegression()
reg.fit(Z_train, y_train)

train_error = mean_squared_error(y_train, reg.predict(Z_train))
val_error = mean_squared_error(y_val, reg.predict(Z_val))

print("Train/Val errors obtained when using new features Z to predict tempature␣
↪y")
print("Training Error:", train_error)
```

```python
print("Validation Error:", val_error)
print("\n*******************************************************************\n")

# Split into training and validation sets.
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=4740)

reg = LinearRegression()
reg.fit(X_train, y_train)

train_error = mean_squared_error(y_train, reg.predict(X_train))
val_error = mean_squared_error(y_val, reg.predict(X_val))

print("Train/Val errors obtained when using original (privacy-leaking) features␣
 ↪X to predict tempature y")
print("Training Error:", train_error)
print("Validation Error:", val_error)
```

```
Train/Val errors obtained when using new features Z to predict tempature y
Training Error: 31.08297194420889
Validation Error: 30.04272420998213

*******************************************************************

Train/Val errors obtained when using original (privacy-leaking) features X to
predict tempature y
Training Error: 16.735639571838544
Validation Error: 16.67841632785604
```

[ ]:

[ ]:

[ ]: