Insurance cost prediction using linear regression

In this assignment we're going to use information like a person's age, sex, BMI, no. of children and smoking habit to predict the price of yearly medical bills. This kind of model is useful for insurance companies to determine the yearly insurance premium for a person. The dataset for this problem is taken from: https://www.kaggle.com/mirichoi0218/insurance

We will create a model with the following steps:

1. Download and explore the dataset
2. Prepare the dataset for training
3. Create a linear regression model
4. Train the model to fit the data
5. Make predictions using the trained model

This assignment builds upon the concepts from the first 2 lectures. It will help to review these Jupyter notebooks:

- PyTorch basics: https://jovian.ml/aakashns/01-pytorch-basics
- Linear Regression: https://jovian.ml/aakashns/02-linear-regression
- Logistic Regression: https://jovian.ml/aakashns/03-logistic-regression
- Linear regression (minimal): https://jovian.ml/aakashns/housing-linear-minimal
- Logistic regression (minimal): https://jovian.ml/aakashns/mnist-logistic-minimal

As you go through this notebook, you will find a **???** in certain places. Your job is to replace the **???** with appropriate code or values, to ensure that the notebook runs properly end-to-end . In some cases, you'll be required to choose some hyperparameters (learning rate, batch size etc.). Try to experiment with the hypeparameters to get the lowest loss.

In [1]:

```
# Uncomment and run the commands below if imports fail
# !conda install numpy pytorch torchvision cpuonly -c pytorch -y
# !pip install matplotlib --upgrade --quiet
!pip install jovian --upgrade --quiet
```

```
WARNING: You are using pip version 20.1; however, version 20.1.1 is available.

You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' command.
```

In [2]:

```
import torch
import jovian
import torchvision
import torch.nn as nn
import pandas as pd
import matplotlib.pyplot as plt
import torch.nn.functional as F
from torchvision.datasets.utils import download_url
from torch.utils.data import DataLoader, TensorDataset, random_split
```

In [3]:

```
project_name='02-insurance-linear-regression' # will be used by jovian.commit
```

Step 1: Download and explore the data

Let us begin by downloading the data. We'll use the `download_url` function from PyTorch to get the data as a CSV (comma-separated values) file.

In [4]:

```
DATASET_URL = "https://hub.jovian.ml/wp-content/uploads/2020/05/insurance.csv"
DATA_FILENAME = "insurance.csv"
```

```
download_url(DATASET_URL, '.')
```

```
Downloading https://hub.jovian.ml/wp-content/uploads/2020/05/insurance.csv to ./insurance.csv
```

To load the dataset into memory, we'll use the `read_csv` function from the `pandas` library. The data will be loaded as a Pandas dataframe. See this short tutorial to learn more: https://data36.com/pandas-tutorial-1-basics-reading-data-files-dataframes-data-selection/

In [5]:

```
dataframe_raw = pd.read_csv(DATA_FILENAME)
dataframe_raw.head()
```

Out[5]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

We're going to do a slight customization of the data, so that you every participant receives a slightly different version of the dataset. Fill in your name below as a string (enter at least 5 characters)

In [ ]:

```
your_name = ??? # at least 5 characters
```

The `customize_dataset` function will customize the dataset slightly using your name as a source of random numbers.

In [ ]:

```
def customize_dataset(dataframe_raw, rand_str):
    dataframe = dataframe_raw.copy(deep=True)
    # drop some rows
    dataframe = dataframe.sample(int(0.95*len(dataframe)), random_state=int(ord(rand_str[0])))
    # scale input
    dataframe.bmi = dataframe.bmi * ord(rand_str[1])/100.
    # scale target
    dataframe.charges = dataframe.charges * ord(rand_str[2])/100.
    # drop column
    if ord(rand_str[3]) % 2 == 1:
        dataframe = dataframe.drop(['region'], axis=1)
    return dataframe
```

In [ ]:

```
dataframe = customize_dataset(dataframe_raw, your_name)
dataframe.head()
```

Let us answer some basic questions about the dataset.

Q: How many rows does the dataset have?

In [ ]:

```
num_rows = ???
print(num_rows)
```

Q: How many columns doe the dataset have

In [ ]:

```
num_cols = ???
print(num_cols)
```

Q: What are the column titles of the input variables?

In [ ]:

```
input_cols = [???]
```

Q: Which of the input columns are non-numeric or categorial variables ?

Hint: `sex` is one of them. List the columns that are not numbers.

In [ ]:

```
categorical_cols = [???]
```

Q: What are the column titles of output/target variable(s)?

In [ ]:

```
output_cols = [???]
```

Q: (Optional) What is the minimum, maximum and average value of the `charges` column? Can you show the distribution of values in a graph? Use this data visualization cheatsheet for referece: https://jovian.ml/aakashns/dataviz-cheatsheet

In [ ]:

```
# Write your answer here
```

Remember to commit your notebook to Jovian after every step, so that you don't lose your work.

In [ ]:

```
jovian.commit(project=project_name, environment=None)
```

Step 2: Prepare the dataset for training

We need to convert the data from the Pandas dataframe into a PyTorch tensors for training. To do this, the first step is to convert it numpy arrays. If you've filled out `input_cols`, `categorial_cols` and `output_cols` correctly, this following function will perform the conversion to numpy arrays.

In [ ]:

```
def dataframe_to_arrays(dataframe):
    # Make a copy of the original dataframe
    dataframe1 = dataframe.copy(deep=True)
    # Convert non-numeric categorical columns to numbers
    for col in categorical_cols:
        dataframe1[col] = dataframe1[col].astype('category').cat.codes
    # Extract input & outupts as numpy arrays
    inputs_array = dataframe1[input_cols].to_numpy()
    targets_array = dataframe1[output_cols].to_numpy()
    return inputs_array, targets_array
```

Read through the Pandas documentation to understand how we're converting categorical variables into numbers.

In [ ]:

```
inputs_array, targets_array = dataframe_to_arrays(dataframe)
inputs_array, targets_array
```

Q: Convert the numpy arrays `inputs_array` and `targets_array` into PyTorch tensors. Make sure that the data type is `torch.float32`.

In [ ]:

```
inputs = ???
targets = ???
```

In [ ]:

```
inputs.dtype, targets.dtype
```

Next, we need to create PyTorch datasets & data loaders for training & validation. We'll start by creating a `TensorDataset`.

In [ ]:

```
dataset = TensorDataset(inputs, targets)
```

Q: Pick a number between `0.1` and `0.2` to determine the fraction of data that will be used for creating the validation set. Then use `random_split` to create training & validation datasets.

In [ ]:

```
val_percent = ??? # between 0.1 and 0.2
val_size = int(num_rows * val_percent)
train_size = num_rows - val_size


train_ds, val_ds = ??? # Use the random_split function to split dataset into 2 parts of the desired len
gth
```

Finally, we can create data loaders for training & validation.

Q: Pick a batch size for the data loader.

In [ ]:

```
batch_size = ???
```

In [ ]:

```
train_loader = DataLoader(train_ds, batch_size, shuffle=True)
val_loader = DataLoader(val_ds, batch_size)
```

Let's look at a batch of data to verify everything is working fine so far.

In [ ]:

```
for xb, yb in train_loader:
    print("inputs:", xb)
    print("targets:", yb)
    break
```

Let's save our work by committing to Jovian.

In [ ]:

```
jovian.commit(project=project_name, environment=None)
```

Step 3: Create a Linear Regression Model

Our model itself is a fairly straightforward linear regression (we'll build more complex models in the next assignment).

In [ ]:

```
input_size = len(input_cols)
output_size = len(output_cols)
```

Q: Complete the class definition below by filling out the constructor (`__init__`), `forward`, `training_step` and `validation_step` methods.

Hint: Think carefully about picking a good loss fuction (it's not cross entropy). Maybe try 2-3 of them and see which one works best. See https://pytorch.org/docs/stable/nn.functional.html#loss-functions

In [ ]:

```
class InsuranceModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = ???                   # fill this (hint: use input_size & output_size defined abov
e)

    def forward(self, xb):
        out = ???                           # fill this
        return out

    def training_step(self, batch):
        inputs, targets = batch
        # Generate predictions
        out = self(inputs)
        # Calcuate loss
        loss = ???                           # fill this
        return loss

    def validation_step(self, batch):
        inputs, targets = batch
        # Generate predictions
        out = self(inputs)
        # Calculate loss
        loss = ???                           # fill this
        return {'val_loss': loss.detach()}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        return {'val_loss': epoch_loss.item()}

    def epoch_end(self, epoch, result, num_epochs):
        # Print result every 20th epoch
        if (epoch+1) % 20 == 0 or epoch == num_epochs-1:
            print("Epoch [{}], val_loss: {:.4f}".format(epoch+1, result['val_loss']))
```

Let us create a model using the `InsuranceModel` class. You may need to come back later and re-run the next cell to reinitialize the model, in case the loss becomes `nan` or `infinity`.

In [ ]:

```
model = InsuranceModel()
```

Let's check out the weights and biases of the model using `model.parameters`.

In [ ]:

```
list(model.parameters())
```

One final commit before we train the model

One final commit before we train the model.

In [ ]:

```
jovian.commit(project=project_name, environment=None)
```

Step 4: Train the model to fit the data

To train our model, we'll use the same `fit` function explained in the lecture. That's the benefit of defining a generic training loop - you can use it for any problem.

In [ ]:

```
def evaluate(model, val_loader):
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        for batch in train_loader:
            loss = model.training_step(batch)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        model.epoch_end(epoch, result, epochs)
        history.append(result)
    return history
```

Q: Use the `evaluate` function to calculate the loss on the validation set before training.

In [ ]:

```
result = ??? # Use the the evaluate function
print(result)
```

We are now ready to train the model. You may need to run the training loop many times, for different number of epochs and with different learning rates, to get a good result. Also, if your loss becomes too large (or `nan`), you may have to re-initialize the model by running the cell `model = InsuranceModel()`. Experiment with this for a while, and try to get to as low a loss as possible.

Q: Train the model 4-5 times with different learning rates & for different number of epochs.

Hint: Vary learning rates by orders of 10 (e.g. `1e-2`, `1e-3`, `1e-4`, `1e-5`, `1e-6`) to figure out what works.

In [ ]:

```
epochs = ??
lr = ??
history1 = fit(epochs, lr, model, train_loader, val_loader)
```

In [ ]:

```
epochs = ??
lr = ??
history2 = fit(epochs, lr, model, train_loader, val_loader)
```

In [ ]:

```
epochs = ??
lr = ??
history3 = fit(epochs, lr, model, train_loader, val_loader)
```

In [ ]:

```
epochs = ???
lr = ???
history4 = fit(epochs, lr, model, train_loader, val_loader)
```

In [ ]:

```
epochs = ???
lr = ???
history5 = fit(epochs, lr, model, train_loader, val_loader)
```

Q: What is the final validation loss of your model?

In [ ]:

```
val_loss = ???
```

Let's log the final validation loss to Jovian and commit the notebook

In [ ]:

```
jovian.log_metrics(val_loss=val_loss)
```

In [ ]:

```
jovian.commit(project=project_name, environment=None)
```

Now scroll back up, re-initialize the model, and try different set of values for batch size, number of epochs, learning rate etc. Commit each experiment and use the "Compare" and "View Diff" options on Jovian to compare the different results.

Step 5: Make predictions using the trained model

Q: Complete the following function definition to make predictions on a single input

In [ ]:

```
def predict_single(input, target, model):
    inputs = input.unsqueeze(0)
    predictions = ???                # fill this
    prediction = predictions[0].detach()
    print("Input:", input)
    print("Target:", target)
    print("Prediction:", prediction)
```

In [ ]:

```
input, target = val_ds[0]
predict_single(input, target, model)
```

In [ ]:

```
input, target = val_ds[10]
predict_single(input, target, model)
```

In [ ]:

```
input, target = val_ds[23]
predict_single(input, target, model)
```

Are you happy with your model's predictions? Try to improve them further.

(Optional) Step 6: Try another dataset & blog about it

While this last step is optional for the submission of your assignment, we highly recommend that you do it. Try to clean up & replicate this notebook (or this one, or this one) for a different linear regression or logistic regression problem. This will help solidify your understanding, and give you a chance to differentiate the generic patters in machine learning from problem-specific details.

Here are some sources to find good datasets:

- https://lionbridge.ai/datasets/10-open-datasets-for-linear-regression/
- https://www.kaggle.com/rtatman/datasets-for-regression-analysis
- https://archive.ics.uci.edu/ml/datasets.php?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table
- https://people.sc.fsu.edu/~jburkardt/datasets/regression/regression.html
- https://archive.ics.uci.edu/ml/datasets/wine+quality
- https://pytorch.org/docs/stable/torchvision/datasets.html

We also recommend that you write a blog about your approach to the problem. Here is a suggested structure for your post (feel free to experiment with it):

- Interesting title & subtitle
- Overview of what the blog covers (which dataset, linear regression or logistic regression, intro to PyTorch)
- Downloading & exploring the data
- Preparing the data for training
- Creating a model using PyTorch
- Training the model to fit the data
- Your thoughts on how to experiment with different hyperparmeters to reduce loss
- Making predictions using the model

As with the previous assignment, you can embed Juptyer notebook cells & outputs from Jovian into your blog.

Don't forget to share your work on the forum: https://jovian.ml/forum/t/share-your-work-here-assignment-2/4931

In [ ]:

```
jovian.commit(project=project_name, environment=None)
jovian.commit(project=project_name, environment=None) # try again, kaggle fails sometimes
```

```
[jovian] Attempting to save notebook..
[jovian] Please enter your API key ( from https://jovian.ml/ ):
API KEY: ········
[jovian] Creating a new project "aakashns/02-insurance-linear-regression"
[jovian] Uploading notebook..
[jovian] Committed successfully! https://jovian.ml/aakashns/02-insurance-linear-regression
```

In [ ]:

```
```