Assignment 1. Music Century Classification

Assignment Responsible: Natalie Lang.

In this assignment, we will build models to predict which **century** a piece of music was released. We will be using the "YearPredictionMSD Data Set" based on the Million Song Dataset. The data is available to download from the UCI Machine Learning Repository. Here are some links about the data:

- https://archive.ics.uci.edu/ml/datasets/yearpredictionmsd
- http://millionsongdataset.com/pages/tasks-demos/#yearrecognition

Note that you are note allowed to import additional packages **(especially not PyTorch)**. One of the objectives is to understand how the training procedure actually operates, before working with PyTorch's autograd engine which does it all for us.

Question 1. Data (21%)

Start by setting up a Google Colab notebook in which to do your work. Since you are working with a partner, you might find this link helpful:

- https://colab.research.google.com/github/googlecolab/colabtools/blob/master/notebooks/colab-github-demo.ipynb

The recommended way to work together is pair coding, where you and your partner are sitting together and writing code together.

To process and read the data, we use the popular `pandas` package for data analysis.

```
import pandas
import numpy as np
import matplotlib.pyplot as plt
```

Now that your notebook is set up, we can load the data into the notebook. The code below provides two ways of loading the data: directly from the internet, or through mounting Google Drive. The first method is easier but slower, and the second method is a bit involved at first, but can save you time later on. You will need to mount Google Drive for later assignments, so we recommend figuring how to do that now.

Here are some resources to help you get started:

- http.://colab.research.google.com/notebooks/io.ipynb

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
if not load_from_drive:
  csv_path = "http://archive.ics.uci.edu/ml/machine-learning-databases/00203/YearPredictic
else:
  from google.colab import drive
  drive.mount('/content/gdrive')
  csv_path = '/content/gdrive/My Drive/YearPredictionMSD.txt.zip' # TODO - UPDATE ME WITH

t_label = ["year"]
x_labels = ["var%d" % i for i in range(1, 91)]
df = pandas.read_csv(csv_path, names=t_label + x_labels)
```

Now that the data is loaded to your Colab notebook, you should be able to display the Pandas DataFrame `df` as a table:

```
df
```

	year	var1	var2	var3	var4	var5	var6	var7
0	2001	49.94357	21.47114	73.07750	8.74861	-17.40628	-13.09905	-25.01202
1	2001	48.73215	18.42930	70.32679	12.94636	-10.32437	-24.83777	8.76630
2	2001	50.95714	31.85602	55.81851	13.41693	-6.57898	-18.54940	-3.27872
3	2001	48.24750	-1.89837	36.29772	2.58776	0.97170	-26.21683	5.05097
4	2001	50.97020	42.20998	67.09964	8.46791	-15.85279	-16.81409	-12.48207
...	...	...	...	...	...	...	...	...
515340	2006	51.28467	45.88068	22.19582	-5.53319	-3.61835	-16.36914	2.12652
515341	2006	49.87870	37.93125	18.65987	-3.63581	-27.75665	-18.52988	7.76108
515342	2006	45.12852	12.65758	-38.72018	8.80882	-29.29985	-2.28706	-18.40424
515343	2006	44.16614	32.38368	-3.34971	-2.49165	-19.59278	-18.67098	8.78428
515344	2005	51.85726	59.11655	26.39436	-5.46030	-20.69012	-19.95528	-6.72771

515345 rows × 91 columns

To set up our data for classification, we'll use the "year" field to represent whether a song was released in the 20-th century. In our case `df["year"]` will be 1 if the year was released after 2000, and 0 otherwise.

```
df["year"] = df["year"].map(lambda x: int(x > 2000))
```

```
df.head(20)
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

	year	var1	var2	var3	var4	var5	var6	var7	
0	1	49.94357	21.47114	73.07750	8.74861	-17.40628	-13.09905	-25.01202	-1
1	1	48.73215	18.42930	70.32679	12.94636	-10.32437	-24.83777	8.76630	-
2	1	50.95714	31.85602	55.81851	13.41693	-6.57898	-18.54940	-3.27872	-
3	1	48.24750	-1.89837	36.29772	2.58776	0.97170	-26.21683	5.05097	-1
4	1	50.97020	42.20998	67.09964	8.46791	-15.85279	-16.81409	-12.48207	-
5	1	50.54767	0.31568	92.35066	22.38696	-25.51870	-19.04928	20.67345	-
6	1	50.57546	33.17843	50.53517	11.55217	-27.24764	-8.78206	-12.04282	-
7	1	48.26892	8.97526	75.23158	24.04945	-16.02105	-14.09491	8.11871	-
8	1	49.75468	33.99581	56.73846	2.89581	-2.92429	-26.44413	1.71392	-
9	1	45.17809	46.34234	-40.65357	-2.47909	1.21253	-0.65302	-6.95536	-1
10	1	39.13076	-23.01763	-36.20583	1.67519	-4.27101	13.01158	8.05718	-
11	1	37.66498	-34.05910	-17.36060	-26.77781	-39.95119	-20.75000	-0.10231	-
12	1	26.51957	-148.15762	-13.30095	-7.25851	17.22029	-21.99439	5.51947	
13	1	37.68491	-26.84185	-27.10566	-14.95883	-5.87200	-21.68979	4.87374	-1
14	0	39.11695	-8.29767	-51.37966	-4.42668	-30.06506	-11.95916	-0.85322	-
15	1	35.05129	-67.97714	-14.20239	-6.68696	-0.61230	-18.70341	-1.31928	-
16	1	33.63129	-96.14912	-89.38216	-12.11699	13.77252	-6.69377	-33.36843	-2
17	0	41.38639	-20.78665	51.80155	17.21415	-36.44189	-11.53169	11.75252	-
18	0	37.45034	11.42615	56.28982	19.58426	-16.43530	2.22457	1.02668	-

Part (a) -- 7%

The data set description text asks us to respect the below train/test split to avoid the "producer effect". That is, we want to make sure that no song from a single artist ends up in both the training and test set.

Explain why it would be problematic to have some songs from an artist in the training set, and other songs from the same artist in the test set. (Hint: Remember that we want our test accuracy to predict how well the model will perform in practice on a song it hasn't learned about.)

```
df_train = df[:463715]
df_test = df[463715:]
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
train_ts = df_train[t_label].to_numpy()
test_xs = df_test[x_labels].to_numpy()
```

```
test_ts = df_test[t_label].to_numpy()

# Write your explanation here
'''The training set can give a measure of the model's success - if the model manages to fi
it is a sign that the model has managed to find rules that can be
also true for new examples to come. To make sure that we evaluate our model correctly we w
 a song it hasn't learned about.
Using some songs from an artist in the training set, and other songs from the same artist
can lead us to a situation where the model will learn specific featues of the singer
such as the quality of the singer's voice, and will predict the decade according to the si
In that situation, we will get a good test accuracy because the singer's songs are in the
but that does not mean the model will perfom the same on different singers and because of
it's problematic to have some songs from an artist in the training set, and other songs
from the same artist in the test set.'''
```

```
'The training set can give a measure of the model's success - if the model manages
to find patterns in the training set that are also true for the test set,\nit is a
sign that the model has managed to find rules that can be\nalso true for new exampl
es to come. To make sure that we evaluate our model correctly we want to know how w
ell the model will perform in practice on\n a song it hasn't learned about.\nUsing
some songs from an artist in the training set, and other songs from the same artist
```

Part (b) -- 7%

It can be beneficial to **normalize** the columns, so that each column (feature) has the *same* mean and standard deviation.

```
feature_means = df_train.mean()[1:].to_numpy() # the [1:] removes the mean of the "year" f
feature_stds  = df_train.std()[1:].to_numpy()

train_norm_xs = (train_xs - feature_means) / feature_stds
test_norm_xs = (test_xs - feature_means) / feature_stds
```

Notice how in our code, we normalized the test set using the *training data means and standard deviations*. This is *not* a bug.

Explain why it would be improper to compute and use test set means and standard deviations. (Hint: Remember what we want to use the test accuracy to measure.)

```
# Write your explanation here
'''The conceptual idea is that test data is supposed to mimic new, previously unseen data
If we normalize the test data separately we can cause data leakage,
meaing we are introducing future information into the training explanatory variables (i.e.
Therefore, we should perform feature normalisation over the training data,
and then perform normalisation on testing data me using the mean and variance of training
In this way, we can test and evaluate whether our model can generalize well to new, unseer
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
variables (i.e. the mean and variance).\nTherefore, we should perform feature norma
```

Part (c) -- 7%

Finally, we'll move some of the data in our training set into a validation set.

Explain why we should limit how many times we use the test set, and that we should use the validation set during the model building process.

```
# shuffle the training set
reindex = np.random.permutation(len(train_xs))
train_xs = train_xs[reindex]
train_norm_xs = train_norm_xs[reindex]
train_ts = train_ts[reindex]

# use the first 50000 elements of `train_xs` as the validation set
train_xs, val_xs           = train_xs[50000:], train_xs[:50000]
train_norm_xs, val_norm_xs = train_norm_xs[50000:], train_norm_xs[:50000]
train_ts, val_ts           = train_ts[50000:], train_ts[:50000]

# Write your explanation here
'''First , we should limit how many times we use the test set becaus we want to evaluate t
If we use the test data and change our training according to the test data we will get goc
to unseen data, hence we sould use the validation data insead.
We use validation set during the model building proccess so we can give an estimate of moc
The validation dataset is different from the test dataset , both datasets are held back fr
but the validation set used to give an unbiased estimate of the skill of the final tuned m
```

```
'First , we should limit how many times we use the test set becaus we want to evalu
ate the model according to unseen data.\nIf we use the test data and change our tra
ining according to the test data we will get good test accuracy but it won't be gen
eralize\nto unseen data, hence we sould use the validation data insead.\nWe use val
idation set during the model building proccess so we can give an estimate of model
```

Part 2. Classification (79%)

We will first build a *classification* model to perform decade classification. These helper functions are written for you. All other code that you write in this section should be vectorized whenever possible (i.e., avoid unnecessary loops).

```
def sigmoid(z):
  return 1 / (1 + np.exp(-z))

def cross_entropy(t, y):
  epsilon = 1e-05
  return -t * np.log(y+epsilon) - (1 - t) * np.log(1 - y+epsilon)

def cost(y, t):
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
  acc = 0
  N = 0
```

```
    for i in range(len(y)):
      N += 1
      if (y[i] >= 0.5 and t[i] == 1) or (y[i] < 0.5 and t[i] == 0):
        acc += 1
    return acc / N
```

Part (a) -- 7%

Write a function `pred` that computes the prediction `y` based on logistic regression, i.e., a single layer with weights `w` and bias `b`. The output is given by:

$$y = \sigma(\mathbf{w}^T\mathbf{x} + b),$$

where the value of y is an estimate of the probability that the song is released in the current century, namely $\mathrm{year} = 1$.

```
def pred(w, b, X):
  """
  Returns the prediction `y` of the target based on the weights `w` and scalar bias `b`.

  Preconditions: np.shape(w) == (90,)
                 type(b) == float
                 np.shape(X) = (N, 90) for some N

  >>> pred(np.zeros(90), 1, np.ones([2, 90]))
  array([0.73105858, 0.73105858]) # It's okay if your output differs in the last decimals
  """
  # Your code goes here
  z=np.dot(X,w)+b
  y=sigmoid(z)
  return y
```

```
pred(np.zeros(90), 1, np.ones([2, 90]))
```

```
    array([0.73105858, 0.73105858])
```

Part (b) -- 7%

Write a function `derivative_cost` that computes and returns the gradients $\frac{\partial\mathcal{L}}{\partial\mathbf{w}}$ and $\frac{\partial\mathcal{L}}{\partial b}$. Here, `X` is the input, `y` is the prediction, and `t` is the true label.

```
def derivative_cost(X, y, t):
  """
  Returns a tuple containing the gradients dLdw and dLdb.

  Precondition: np.shape(X) == (N, 90) for some N
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
  Postcondition: np.shape(dLdw) = (90,)
                 type(dLdb) = float
```

```
            type(dLdb) = float
    """
    # Your code goes here
    N=len(y)
    epsilon=1e-05
    dLdy=(-t/(y+epsilon)+(1-t)/(1-y+epsilon))/N
    sig_d=y*(1-y)
    dydb=sig_d
    dLdb=np.dot(dydb,dLdy)
    dydw=sig_d*X.T
    dLdw=np.dot(dydw,dLdy)
    return (dLdw,dLdb)
```

Explenation on Gradients

Add here an explaination on how the gradients are computed:

Write your explanation here. Use Latex to write mathematical expressions. Here is a brief tutorial on latex for notebooks.

We calculate $\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}$ using the chain rule :

$L(w,b,X,t) = -\frac{1}{N}\sum_{n=1}^{N} t_n \log(y_n) + (1-t_n)\log(1-y_n) = -\frac{1}{N}(t\cdot\log(y) + (1-t)$

So $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial w}$, and $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial b}$

where:

$\frac{\partial L}{\partial y} = -\frac{1}{N}(\frac{t}{y} - \frac{1-t}{1-y}) = -\frac{1}{N}\frac{t-y}{y\cdot(1-y)}$

$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial b} = \sigma'(z)\cdot 1 = \sigma(z)(1-\sigma(z)) = y(1-y)$

$\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \sigma'(z)X = (\sigma(z\cdot)(1-\sigma(z)))X = (y\cdot(1-y))X$

Part (c) -- 7%

We can check that our derivative is implemented correctly using the finite difference rule. In 1D, the finite difference rule tells us that for small h, we should have

$$\frac{f(x+h)-f(x)}{h} \approx f'(x)$$

Show that $\frac{\partial \mathcal{L}}{\partial b}$ is implement correctly by comparing the result from `derivative_cost` with the empirical cost derivative computed using the above numerical approximation.

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
def compute_analytical_deriviate(w,b,t,X,parameter_to_check,h=1e-05):
  y= pred(w, b, X)
```

```
    L=cost(y,t)
    if parameter_to_check=='b':
      y_plus=pred(w,b+h,X)
      L_plus=cost(y_plus,t)
      deriviate=(L_plus-L)/h

    elif parameter_to_check=='w':
      L_plus=np.zeros(len(w))
      deriviate=np.zeros(len(w))
      for i in range(len(w)):
        # Compute cost of w[i] + h
        w_plus = np.copy(w)
        w_plus[i] = w_plus[i] + h
        y_plus=pred(w_plus,b,X)
        L_plus[i]=cost(y_plus,t)
        deriviate[i] = (L_plus[i]-L)/h

    else:
      raise Exception('parameter_to_check: {} is not define - has to be b or w'.format(param

    return deriviate
```

```
#example num 1:
w=np.zeros(90)
h=1e-05
b=1
t=np.array([1,1])
X=np.ones([2, 90])
y= pred(w, b, X)
dw,db=derivative_cost(X,y,t)
analytical_deriviate=compute_analytical_deriviate(w,b,t,X,'b')

print("The analytical results is -", analytical_deriviate)
print("The algorithm results is - ",db)
```

```
The analytical results is - -0.2689367595898329
The algorithm results is -  -0.2689377426259042
```

We got that our derivative is implemented correctly using the finite difference rule.

Part (d) -- 7%

Show that $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ is implement correctly.

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
# up to you to figure out how/why, and how to modify the code
```

```
analytical_deriviate=compute_analytical_deriviate(w,b,t,X,'w')
print("The analytical results is -", analytical_deriviate,'\n')
print("The algorithm results is - ", dw)
```

```
The analytical results is - [-0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.2689
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676
 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676 -0.26893676]

The algorithm results is -  [-0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.2689
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774
 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774 -0.26893774]
```

Part (e) -- 7%

Now that you have a gradient function that works, we can actually run gradient descent. Complete the following code that will run stochastic: gradient descent training:

```
def run_gradient_descent(w0, b0, mu=0.1, batch_size=100, max_iters=100):
  """Return the values of (w, b) after running gradient descent for max_iters.
  We use:
    - train_norm_xs and train_ts as the training set
    - val_norm_xs and val_ts as the test set
    - mu as the learning rate
    - (w0, b0) as the initial values of (w, b)

  Precondition: np.shape(w0) == (90,)
                type(b0) == float
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
                 type(b) == float
  """
  w = w0
  b = b0
  iter = 0

  l=[]
  accuracies=[]
  losses=[]

  while iter < max_iters:
    # shuffle the training set (there is code above for how to do this)
    reindex = np.random.permutation(len(train_norm_xs))
    train_norm_xs_local = train_norm_xs[reindex]
    train_ts_local = train_ts[reindex]


    for i in range(0, len(train_norm_xs), batch_size): # iterate over each minibatch
      # minibatch that we are working with:
      X = train_norm_xs[i:(i + batch_size)]
      t = train_ts[i:(i + batch_size), 0]

      # since len(train_norm_xs) does not divide batch_size evenly, we will skip over
      # the "last" minibatch
      if np.shape(X)[0] != batch_size:
        continue

      # compute the prediction
      y=pred(w,b,X)
      # update w and b
      dw,db=derivative_cost(X,y,t)
      w-=mu*dw
      b-=mu*db
      # increment the iteration count
      iter += 1
      # compute and print the *validation* loss and accuracy
      if (iter % 10 == 0):
        y_val=pred(w,b,val_norm_xs)
        val_cost = cost(y_val,val_ts.squeeze())
        val_acc = get_accuracy(y_val,val_ts.squeeze())
        print("Iter %d. [Val Acc %.0f%%, Loss %f]" % (
                iter, val_acc * 100, val_cost))
        losses.append(val_cost)
        accuracies.append(val_acc)
      if iter >= max_iters:
        break

      # Think what parameters you should return for further use
      # we decided to return the losses according to iteration so we can se the loss conve
  return w,b,losses
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

Part (f) -- 7%

Call `run_gradient_descent` with the weights and biases all initialized to zero. Show that if the learning rate μ is too small, then convergence is slow. Also, show that if μ is too large, then the optimization algorirthm does not converge. The demonstration should be made using plots showing these effects.

```
w0 = np.random.randn(90)
b0 = np.random.randn(1)[0]

# Write your code here
mus=[0.02,0.5,6]
for mu in mus:
  print("mu=",mu,":")
  w,b,losses=run_gradient_descent(w0,b0,mu)
  print("-------------------------")
  plt.plot(range(0,100,10),losses)

plt.legend(mus,loc='upper right')
plt.xlabel("number of iteration")
plt.ylabel("Loss")
plt.title("The affect of mu on the convergence")
```

```
mu= 0.02 :
Iter 10. [Val Acc 47%, Loss 2.919458]
Iter 20. [Val Acc 48%, Loss 2.884831]
Iter 30. [Val Acc 48%, Loss 2.854329]
Iter 40. [Val Acc 48%, Loss 2.823747]
Iter 50. [Val Acc 48%, Loss 2.796757]
Iter 60. [Val Acc 48%, Loss 2.766477]
Iter 70. [Val Acc 48%, Loss 2.735379]
Iter 80. [Val Acc 48%, Loss 2.704835]
Iter 90. [Val Acc 49%, Loss 2.676411]
Iter 100. [Val Acc 49%, Loss 2.645081]
-------------------------
mu= 0.5 :
Iter 10. [Val Acc 53%, Loss 1.984112]
Iter 20. [Val Acc 57%, Loss 1.626597]
Iter 30. [Val Acc 60%, Loss 1.381379]
Iter 40. [Val Acc 62%, Loss 1.171015]
Iter 50. [Val Acc 64%, Loss 1.052483]
Iter 60. [Val Acc 65%, Loss 0.952409]
Iter 70. [Val Acc 66%, Loss 0.870553]
Iter 80. [Val Acc 68%, Loss 0.793330]
```

Explain and discuss your results here: We can see in the graph, if the learning rate is set too low, for example $\mu = 0.02$, the training progress very slowly because in each epoch we are making very tiny updates to the weights. And if the learning rate is set too high, for example $\mu = 6$ it cause undesirable divergent behavior in our loss function because the steps do oscillation and do not converge to the local minima.

For regular learning rate such as $\mu = 0.5$ we can tell that we got covergence and the covergence is not too slow.

```
Iter 90. [Val Acc 65%, Loss 1.599089]
```

Part (g) -- 7%

Find the optimial value of $\mathbf{w}$ and b using your code. Explain how you chose the learning rate μ and the batch size. Show plots demostrating good and bad behaviours.

```
w0 = np.random.randn(90)
b0 = np.random.randn(1)[0]

# Write your code here
mu=0.5
batches=[10,70,100,150,420]
for batch_size in batches:
  print("batch_size=",batch_size,":")
  w,b,losses=run_gradient_descent(w0,b0,mu,batch_size)
  print("-------------------------")
  plt.plot(range(0,100,10),losses)

plt.legend(batches,loc='upper right')
```

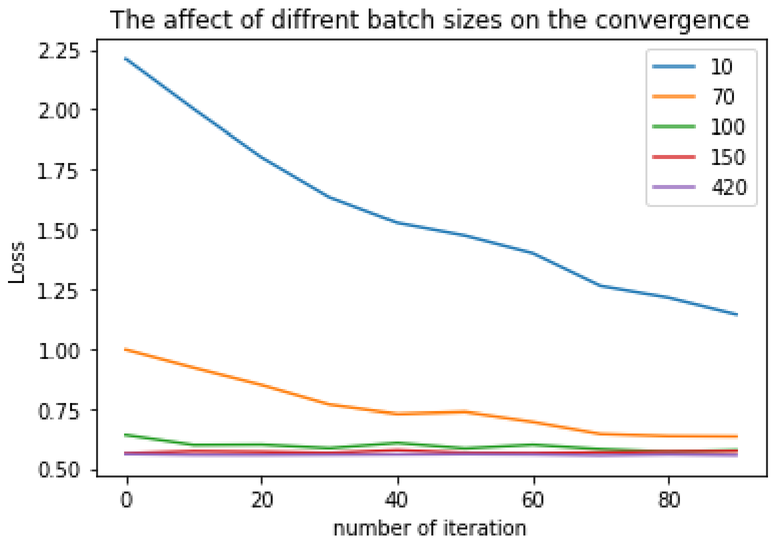
Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
plt.title("The affect of diffrent batch sizes on the convergence")
```

```
batch_size= 10 :
Iter 10. [Val Acc 57%, Loss 2.211221]
Iter 20. [Val Acc 59%, Loss 2.001843]
Iter 30. [Val Acc 61%, Loss 1.800888]
Iter 40. [Val Acc 61%, Loss 1.634075]
Iter 50. [Val Acc 63%, Loss 1.527455]
Iter 60. [Val Acc 63%, Loss 1.473764]
Iter 70. [Val Acc 64%, Loss 1.399950]
Iter 80. [Val Acc 65%, Loss 1.263219]
Iter 90. [Val Acc 66%, Loss 1.213752]
Iter 100. [Val Acc 67%, Loss 1.143388]
-------------------------
batch_size= 70 :
Iter 10. [Val Acc 68%, Loss 0.995662]
Iter 20. [Val Acc 68%, Loss 0.920116]
Iter 30. [Val Acc 68%, Loss 0.848070]
Iter 40. [Val Acc 68%, Loss 0.766348]
Iter 50. [Val Acc 69%, Loss 0.727583]
Iter 60. [Val Acc 67%, Loss 0.736113]
Iter 70. [Val Acc 69%, Loss 0.694954]
Iter 80. [Val Acc 70%, Loss 0.644683]
Iter 90. [Val Acc 70%, Loss 0.635926]
Iter 100. [Val Acc 70%, Loss 0.634339]
-------------------------
batch_size= 100 :
Iter 10. [Val Acc 70%, Loss 0.640338]
Iter 20. [Val Acc 72%, Loss 0.598720]
Iter 30. [Val Acc 71%, Loss 0.600192]
Iter 40. [Val Acc 72%, Loss 0.585398]
Iter 50. [Val Acc 70%, Loss 0.607213]
Iter 60. [Val Acc 72%, Loss 0.584346]
Iter 70. [Val Acc 71%, Loss 0.599661]
Iter 80. [Val Acc 72%, Loss 0.581148]
Iter 90. [Val Acc 73%, Loss 0.570571]
Iter 100. [Val Acc 73%, Loss 0.576832]
-------------------------
batch_size= 150 :
Iter 10. [Val Acc 73%, Loss 0.563775]
Iter 20. [Val Acc 72%, Loss 0.572395]
Iter 30. [Val Acc 72%, Loss 0.569916]
Iter 40. [Val Acc 73%, Loss 0.565218]
Iter 50. [Val Acc 73%, Loss 0.576890]
Iter 60. [Val Acc 72%, Loss 0.565849]
Iter 70. [Val Acc 73%, Loss 0.564577]
Iter 80. [Val Acc 73%, Loss 0.567350]
Iter 90. [Val Acc 72%, Loss 0.571153]
Iter 100. [Val Acc 73%, Loss 0.573232]
-------------------------
batch_size= 420 :
Iter 10. [Val Acc 73%, Loss 0.560913]
Iter 20. [Val Acc 73%, Loss 0.557436]
Iter 30. [Val Acc 73%, Loss 0.557078]
Iter 40. [Val Acc 73%, Loss 0.557957]
Iter 50. [Val Acc 73%, Loss 0.558986]
Iter 60. [Val Acc 73%, Loss 0.560431]
Iter 70. [Val Acc 74%, Loss 0.558916]
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
Iter 100. [Val Acc 74%, Loss 0.556074]
-------------------------
Text(0.5, 1.0, 'The affect of diffrent batch sizes on the convergence')
```



Explain and discuss your results here: We chose $\mu = 0.5$ because what we saw in the last saction (we got covergence and the covergence is not too slow). We can tell that the bigger the batch size we get fast convergence, which make sense because with bigger batches we need much fewer updates for the same accuracy.

However, even if it seems good to use a great batch size choosing too large of a vtch size will lead to poor generalization. hence 100 or even 70 will be good for our model.

Part (h) -- 15%

Using the values of `w` and `b` from part (g), compute your training accuracy, validation accuracy, and test accuracy. Are there any differences between those three values? If so, why?

```
# Write your code here

best_w,best_b,_=run_gradient_descent(w0,b0,0.5,150)
train_ys=pred(best_w,best_b,train_norm_xs)
test_ys=pred(best_w,best_b,test_norm_xs)
val_ys=pred(best_w,best_b,val_norm_xs)


train_acc = get_accuracy(train_ys,train_ts)
val_acc =  get_accuracy(val_ys,val_ts)
test_acc =  get_accuracy(test_ys,test_ts)

print('train_acc = ', train_acc, ' val_acc = ', val_acc, ' test_acc = ', test_acc)
```

```
    Iter 10. [Val Acc 73%, Loss 0.560163]
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
    Iter 50. [Val Acc 73%, Loss 0.576486]
    Iter 60. [Val Acc 73%, Loss 0.564452]
```

```
Iter 70. [Val Acc 73%, Loss 0.563532]
Iter 80. [Val Acc 73%, Loss 0.566486]
Iter 90. [Val Acc 72%, Loss 0.571107]
Iter 100. [Val Acc 73%, Loss 0.572368]
train_acc =  0.7220961289776778  val_acc =  0.72722  test_acc =  0.7191555297307767
```

Explain and discuss your results here: The train accuracy is a little bit better then the test_accuracy because the test data is an unseen data so it is very likely have lower accuracy on an unseen test dataset. The differnce here are not so great, it is a good sign it mean that the model is well generalized.

Val accuracy should be some where between test and train accuracy - it is unseen data but we use the validation set for model selection so we did choose our parameters so the val_acc will be as good as the train_acc

Part (i) -- 15%

Writing a classifier like this is instructive, and helps you understand what happens when we train a model. However, in practice, we rarely write model building and training code from scratch. Instead, we typically use one of the well-tested libraries available in a package.

Use `sklearn.linear_model.LogisticRegression` to build a linear classifier, and make predictions about the test set. Start by reading the API documentation here.

Compute the training, validation and test accuracy of this model.

```
import sklearn.linear_model

model = sklearn.linear_model.LogisticRegression()
model.fit(train_norm_xs,train_ts)
train_ys=model.predict(train_norm_xs)
test_ys=model.predict(test_norm_xs)
val_ys=model.predict(val_norm_xs)

train_acc = get_accuracy(train_ys,train_ts)
val_acc =  get_accuracy(val_ys,val_ts)
test_acc =  get_accuracy(test_ys,test_ts)

print('train_acc = ', train_acc, ' val_acc = ', val_acc, ' test_acc = ', test_acc)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversi
  y = column_or_1d(y, warn=True)
train_acc =  0.7322407937831599  val_acc =  0.7366  test_acc =  0.726593066046872
```

to PDF

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
!apt get install texlive texlive xetex texlive latex extra pandoc
!pip install pypandoc
  Preparing to unpack .../45-lipa_2%3a1.3-20_all.deb ...
```

```
Unpacking tipa (2:1.3-20) ...
Selecting previously unselected package texlive-xetex.
Preparing to unpack .../46-texlive-xetex_2017.20180305-1_all.deb ...
Unpacking texlive-xetex (2017.20180305-1) ...
Setting up libgs9-common (9.26~dfsg+0-0ubuntu0.18.04.14) ...
Setting up libkpathsea6:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up libjs-jquery (3.2.1-1) ...
Setting up libtexlua52:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up fonts-droid-fallback (1:6.0.1r16-1.1) ...
Setting up libsynctex1:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up libptexenc1:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up tex-common (6.09) ...
update-language: texlive-base not installed and configured, doing nothing!
Setting up poppler-data (0.4.8-2) ...
Setting up tex-gyre (20160520-1) ...
Setting up preview-latex-style (11.91-1ubuntu1) ...
Setting up fonts-texgyre (20160520-1) ...

Setting up fonts-noto-mono (20171026-2) ...
Setting up fonts-lato (2.0-2) ...
Setting up libcupsfilters1:amd64 (1.20.2-0ubuntu3.1) ...
Setting up libcupsimage2:amd64 (2.2.7-1ubuntu2.8) ...
Setting up libjbig2dec0:amd64 (0.13-6) ...
Setting up ruby-did-you-mean (1.2.0-2) ...
Setting up t1utils (1.41-2) ...
Setting up ruby-net-telnet (0.1.1-2) ...
Setting up libijs-0.35:amd64 (0.35-13) ...
Setting up rubygems-integration (1.11) ...
Setting up libpotrace0 (1.14-2) ...
Setting up javascript-common (11) ...
Setting up ruby-minitest (5.10.3-1) ...
Setting up libzzip-0-13:amd64 (0.13.62-3.1ubuntu0.18.04.1) ...
Setting up libgs9:amd64 (9.26~dfsg+0-0ubuntu0.18.04.14) ...
Setting up libtexluajit2:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up fonts-lmodern (2.004.5-3) ...
Setting up ruby-power-assert (0.3.0-1) ...
Setting up texlive-binaries (2017.20170613.44572-8ubuntu0.1) ...
update-alternatives: using /usr/bin/xdvi-xaw to provide /usr/bin/xdvi.bin (xdvi.bi
update-alternatives: using /usr/bin/bibtex.original to provide /usr/bin/bibtex (bi
Setting up texlive-base (2017.20180305-1) ...
mktexlsr: Updating /var/lib/texmf/ls-R-TEXLIVEDIST...
mktexlsr: Updating /var/lib/texmf/ls-R-TEXMFMAIN...
mktexlsr: Updating /var/lib/texmf/ls-R...
mktexlsr: Done.
tl-paper: setting paper size for dvips to a4: /var/lib/texmf/dvips/config/config-p
tl-paper: setting paper size for dvipdfmx to a4: /var/lib/texmf/dvipdfmx/dvipdfmx-
tl-paper: setting paper size for xdvi to a4: /var/lib/texmf/xdvi/XDvi-paper
tl-paper: setting paper size for pdftex to a4: /var/lib/texmf/tex/generic/config/p
Setting up texlive-fonts-recommended (2017.20180305-1) ...
Setting up texlive-plain-generic (2017.20180305-2) ...
Setting up texlive-latex-base (2017.20180305-1) ...
Setting up lmodern (2.004.5-3) ...
Setting up texlive-latex-recommended (2017.20180305-1) ...
Setting up texlive-pictures (2017.20180305-1) ...
Setting up tipa (2:1.3-20) ...
Regenerating '/var/lib/texmf/fmtutil.cnf-DEBIAN'... done.
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
from google.colab import drive
drive.mount('/content/drive')

     Mounted at /content/drive

!cp /content/drive/My\ Drive/Colab\ Notebooks/Assignment11.ipynb ./


!jupyter nbconvert --to PDF "Assignment11.ipynb"

     (/usr/share/texlive/texmf-dist/tex/latex/oberdiek/rerunfilecheck.sty))
     (/usr/share/texlive/texmf-dist/tex/latex/tools/longtable.sty)
     (/usr/share/texlive/texmf-dist/tex/latex/booktabs/booktabs.sty)
     (/usr/share/texlive/texmf-dist/tex/latex/enumitem/enumitem.sty)
     (/usr/share/texlive/texmf-dist/tex/generic/ulem/ulem.sty)
     (/usr/share/texlive/texmf-dist/tex/latex/jknapltx/mathrsfs.sty)
     No file notebook.aux.
     (/usr/share/texlive/texmf-dist/tex/latex/base/ts1cmr.fd)
     (/usr/share/texlive/texmf-dist/tex/latex/psnfss/t1ppl.fd)
     ABD: EveryShipout initializing macros
     (/usr/share/texlive/texmf-dist/tex/latex/caption/ltcaption.sty)
     *geometry* driver: auto-detecting
     *geometry* detected driver: xetex
     *geometry* verbose mode - [ preamble ] result:
     * driver: xetex
     * paper: <default>
     * layout: <same size as paper>
     * layoutoffset:(h,v)=(0.0pt,0.0pt)
     * modes:
     * h-part:(L,W,R)=(72.26999pt, 469.75502pt, 72.26999pt)
     * v-part:(T,H,B)=(72.26999pt, 650.43001pt, 72.26999pt)
     * \paperwidth=614.295pt
     * \paperheight=794.96999pt
     * \textwidth=469.75502pt
     * \textheight=650.43001pt
     * \oddsidemargin=0.0pt
     * \evensidemargin=0.0pt
     * \topmargin=-37.0pt
     * \headheight=12.0pt
     * \headsep=25.0pt
     * \topskip=11.0pt
     * \footskip=30.0pt
     * \marginparwidth=59.0pt
     * \marginparsep=10.0pt
     * \columnsep=10.0pt
     * \skip\footins=10.0pt plus 4.0pt minus 2.0pt
     * \hoffset=0.0pt
     * \voffset=0.0pt
     * \mag=1000
     * \@twocolumnfalse
     * \@twosidefalse
     * \@mparswitchfalse
     * \@reversemarginfalse
     * (1in=72.27pt=25.4mm, 1cm=28.453pt)

     (/usr/share/texlive/texmf-dist/tex/latex/ucs/ucsencs.def)
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
     Package hyperref Warning: Rerun to get /PageLabels entry.
```

```
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/omlzplm.fd)
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/omszplm.fd)
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/omxzplm.fd)
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/ot1zplm.fd)
(/usr/share/texlive/texmf-dist/tex/latex/jknapltx/ursfs.fd)

LaTeX Warning: No \author given.
```

This parts helps by checking if the code worked. Check if you get similar results, if not repair your code