Fundamentals of Machine Learning (1618003)

Semester-1

Post Graduate Diploma in Data Science

Prepared By:

Krima Doshi(20137680015)

Course Coordinator:

Prof. Mahesh Panchal

Designation

GTU-GSET

Gujarat Techological University

Graduate School of Engineering and Technology

December, 2020 to March, 2021

Assignment 2

March 24, 2021

Contents

List of Figures

1 Project Aim

Prediction of the release year of a song from audio features. Songs are mostly western, commercial tracks ranging from 1922 to 2011, with a peak in the year 2000s.

2 Data description

Link to dataset: https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD

Number of Instances: 515345
Number of Attributes: 90
Missing Values: None
Associated Tasks: Regression

Training Data: 463,715
Testing Data: 51,630

90 attributes
12 = timbre average
78 = timbre covariance

The first value is the year (target), ranging from 1922 to 2011. Features extracted from the 'timbre' features from The Echo Nest API. We take the average and covariance over all 'segments', each segment being described by a 12-dimensional timbre vector.

Timbre are the ways used to describe the sound, so words such as Light, Flat, Smooth, Smoky, Breathy, Rough, and so on are what you use to distinguish one sound from another. How you recognize the different sounds or voices you hear is attributed to the timbre.

Echo Nest classifies the timbre as: the first dimension represents the average loudness of the segment; second emphasizes brightness; third is more closely correlated to the flatness of a sound; fourth to sounds with a stronger attack; etc

3 Work Environment

Anaconda, jupyter-notebook, python 3.9, tensorflow, keras and scikit-learn are the major modules used.

Linear Regression has been used with a 8GB computer on a 64 bit OS with NVIDIA graphics card.

Neural Network has been applied on a 64 GB RAM GPU on a 64 bit OS with Quadro P400/PCIe/SSE2.

You can setup the environment on anaconda with the following environment.yml file:

```
name: mlenv
channels:
  - pytorch
  - conda-forge
  - anaconda
  - defaults
dependencies:
  - _tflow_select=2.1.0=gpu
  - absl-py=0.12.0=py38h06a4308_0
  - aiohttp=3.6.3=py38h7b6447c_0
  - anyio=2.2.0=py38h578d9bd_0
  - argon2-cffi=20.1.0=py38h25fe258_2
  - astunparse=1.6.3=py_0
  - async-timeout=3.0.1=py38_0
  - async_generator=1.10=py_0
  - attrs=20.2.0=py_0
  - babel=2.9.0=pyhd3deb0d_0
  - backcall=0.2.0=pyh9f0ad1d_0
  - backports=1.0=py_2
  - backports.functools_lru_cache=1.6.1=py_0
  - blas=1.0=mkl
  - bleach=3.3.0=pyh44b312d_0
  - blinker=1.4=py38_0
  - brotlipy=0.7.0=py38h7b6447c_1000
  - c-ares=1.17.1=h27cfd23_0
  - ca-certificates=2020.12.5=ha878542_0
  - cachetools=4.1.1=py_0
  - certifi=2020.12.5=py38h578d9bd_1
  - cffi=1.14.3=py38he30daa8_0
  - chardet=3.0.4=py38_1003
  - click=7.1.2=py_0
  - coverage=5.3=py38h7b6447c_0
  - cryptography=3.1.1=py38h1ba5d50_0
  - cudatoolkit=10.1.243=h6bb024c_0
  - cudnn=7.6.5=cuda10.1_0
  - cupti=10.1.168=0
  - cycler=0.10.0=py_2
  - cython=0.29.21=py38he6710b0_0
  - dbus=1.13.18=hb2f20db_0
  - decorator=4.4.2=py_0
  - defusedxml=0.7.1=pyhd8ed1ab_0
```

```
  - entrypoints=0.3=pyhd8ed1ab_1003
  - expat=2.2.10=he6710b0_2
  - faiss-gpu=1.7.0=py3.8_hf4b0e32_0_cuda10.1
  - fontconfig=2.13.1=he4413a7_1000
  - freetype=2.10.4=h7ca028e_0
  - gast=0.4.0=py_0
  - glib=2.67.4=h36276a3_1
  - google-auth=1.22.1=py_0
  - google-auth-oauthlib=0.4.1=py_2
  - google-pasta=0.2.0=py_0
  - grpcio=1.36.1=py38h2157cd5_1
  - gst-plugins-base=1.14.0=hbbd80ab_1
  - gstreamer=1.14.0=h28cd5cc_2
  - h5py=2.10.0=py38hd6299e0_1
  - hdf5=1.10.6=hb1b8bf9_0
  - icu=58.2=hf484d3e_1000
  - idna=2.10=py_0
  - importlib-metadata=2.0.0=py_1
  - intel-openmp=2020.2=254
  - ipykernel=5.5.0=py38h81c977d_1
  - ipython=7.21.0=py38h81c977d_0
  - ipython_genutils=0.2.0=py_1
  - jedi=0.18.0=py38h578d9bd_2
  - jinja2=2.11.3=pyh44b312d_0
  - joblib=0.17.0=py_0
  - jpeg=9d=h36c2ea0_0
  - json5=0.9.5=pyh9f0ad1d_0
  - jsonschema=3.2.0=pyhd8ed1ab_3
  - jupyter-packaging=0.7.12=pyhd8ed1ab_0
  - jupyter_client=6.1.12=pyhd8ed1ab_0
  - jupyter_core=4.7.1=py38h578d9bd_0
  - jupyter_server=1.4.1=py38h578d9bd_0
  - jupyterlab=3.0.12=pyhd8ed1ab_0
  - jupyterlab_pygments=0.1.2=pyh9f0ad1d_0
  - jupyterlab_server=2.3.0=pyhd8ed1ab_0
  - keras=2.4.3=py_0
  - keras-preprocessing=1.1.2=pyhd3eb1b0_0
  - kiwisolver=1.3.1=py38h82cb98a_0
  - ld_impl_linux-64=2.33.1=h53a641e_7
  - libedit=3.1.20191231=h14c3975_1
  - libfaiss=1.7.0=hbf11de5_0_cuda10.1
  - libffi=3.3=he6710b0_2
  - libgcc-ng=9.1.0=hdf63c60_0
  - libgfortran-ng=7.3.0=hdf63c60_0
  - libpng=1.6.37=h21135ba_2
  - libprotobuf=3.14.0=h8c45485_0
  - libsodium=1.0.18=h36c2ea0_1
  - libstdcxx-ng=9.1.0=hdf63c60_0
  - libtiff=4.0.10=hc3755c2_1005
  - libuuid=2.32.1=h14c3975_1000
  - libxcb=1.13=h14c3975_1002
  - libxml2=2.9.10=hb55368b_3
  - lz4-c=1.9.2=he1b5a44_3
  - markdown=3.3.2=py38_0
  - markupsafe=1.1.1=py38h8df0ef7_2
  - matplotlib=3.3.4=py38h578d9bd_0
  - matplotlib-base=3.3.4=py38h62a2d02_0
  - mistune=0.8.4=py38h25fe258_1002
```

```
  - mkl=2020.2=256
  - mkl-service=2.3.0=py38he904b0f_0
  - mkl_fft=1.3.0=py38h54f3939_0
  - mkl_random=1.1.1=py38h0573a6f_0
  - multidict=4.7.6=py38h7b6447c_1
  - nbclassic=0.2.6=pyhd8ed1ab_0
  - nbclient=0.5.3=pyhd8ed1ab_0
  - nbconvert=6.0.7=py38h578d9bd_3
  - nbformat=5.1.2=pyhd8ed1ab_1
  - ncurses=6.2=he6710b0_1
  - nest-asyncio=1.4.3=pyhd8ed1ab_0
  - notebook=6.3.0=py38h578d9bd_0
  - numpy=1.19.2=py38h54aff64_0
  - numpy-base=1.19.2=py38hfa32c7d_0
  - oauthlib=3.1.0=py_0
  - olefile=0.46=pyh9f0ad1d_1
  - openssl=1.1.1j=h27cfd23_0
  - opt_einsum=3.1.0=py_0
  - packaging=20.9=pyh44b312d_0
  - pandas=1.1.3=py38he6710b0_0
  - pandoc=2.12=h7f98852_0
  - pandocfilters=1.4.2=py_1
  - parso=0.8.1=pyhd8ed1ab_0
  - pcre=8.44=he1b5a44_0
  - pexpect=4.8.0=pyh9f0ad1d_2
  - pickleshare=0.7.5=py_1003
  - pillow=6.2.1=py38h6b7be26_0
  - pip=20.2.4=py38_0
  - prometheus_client=0.9.0=pyhd3deb0d_0
  - prompt-toolkit=3.0.18=pyha770c72_0
  - protobuf=3.14.0=py38h2531618_1
  - pthread-stubs=0.4=h36c2ea0_1001
  - ptyprocess=0.7.0=pyhd3deb0d_0
  - pyasn1=0.4.8=py_0
  - pyasn1-modules=0.2.8=py_0
  - pycparser=2.20=py_2
  - pygments=2.8.1=pyhd8ed1ab_0
  - pyjwt=1.7.1=py38_0
  - pyopenssl=19.1.0=py_1
  - pyparsing=2.4.7=pyh9f0ad1d_0
  - pyqt=5.9.2=py38h05f1152_4
  - pyrsistent=0.17.3=py38h25fe258_1
  - pysocks=1.7.1=py38_0
  - python=3.8.5=h7579374_1
  - python-dateutil=2.8.1=py_0
  - python-flatbuffers=1.12=pyhd3eb1b0_0
  - python_abi=3.8=1_cp38
  - pytz=2021.1=pyhd8ed1ab_0
  - pyyaml=5.3.1=py38h8df0ef7_1
  - pyzmq=19.0.2=py38ha71036d_2
  - qt=5.9.7=h5867ecd_1
  - readline=8.0=h7b6447c_0
  - requests=2.24.0=py_0
  - requests-oauthlib=1.3.0=py_0
  - rsa=4.6=py_0
  - scikit-learn=0.23.2=py38h0573a6f_0
  - scipy=1.6.1=py38h91f5cce_0
  - send2trash=1.5.0=py_0
```

```
  - setuptools=50.3.0=py38hb0f4dca_1
  - sip=4.19.13=py38he6710b0_0
  - six=1.15.0=py_0
  - sniffio=1.2.0=py38h578d9bd_1
  - sqlite=3.33.0=h62c20be_0
  - tensorboard=2.4.0=pyhc547734_0
  - tensorboard-plugin-wit=1.6.0=py_0
  - tensorflow=2.4.1=gpu_py38h8a7d6ce_0
  - tensorflow-base=2.4.1=gpu_py38h29c2da4_0
  - tensorflow-estimator=2.4.1=pyheb71bc4_0
  - tensorflow-gpu=2.4.1=h30adc30_0
  - termcolor=1.1.0=py38_1
  - terminado=0.9.3=py38h578d9bd_0
  - testpath=0.4.4=py_0
  - threadpoolctl=2.1.0=pyh5ca1d4c_0
  - tk=8.6.10=hbc83047_0
  - tornado=6.1=py38h25fe258_0
  - traitlets=5.0.5=py_0
  - urllib3=1.25.11=py_0
  - wcwidth=0.2.5=pyh9f0ad1d_2
  - webencodings=0.5.1=py_1
  - werkzeug=1.0.1=py_0
  - wheel=0.35.1=py_0
  - wrapt=1.12.1=py38h7b6447c_1
  - xorg-libxau=1.0.9=h14c3975_0
  - xorg-libxdmcp=1.1.3=h516909a_0
  - xz=5.2.5=h7b6447c_0
  - yaml=0.2.5=h516909a_0
  - yarl=1.6.2=py38h7b6447c_0
  - zeromq=4.3.4=h2531618_0
  - zipp=3.3.1=py_0
  - zlib=1.2.11=h7b6447c_3
  - zstd=1.4.5=h9ceee32_0
prefix: /home/gtu_hpc/anaconda3/envs/tensor
```

4 Algorithm Selection

The data contains one column for year(integer datatype) and 90 columns for timbre values(float datatype). It looks as follows:

	0	1	2	3	4	5	6	7	8	9	...	81	82	83	84
0	2001	49.94357	21.47114	73.07750	8.74861	-17.40628	-13.09905	-25.01202	-12.23257	7.83089	...	13.01620	-54.40548	58.99367	15.37344
1	2001	48.73215	18.42930	70.32679	12.94636	-10.32437	-24.83777	8.76630	-0.92019	18.76548	...	5.66812	-19.68073	33.04964	42.87836
2	2001	50.95714	31.85602	55.81851	13.41693	-6.57898	-18.54940	-3.27872	-2.35035	16.07017	...	3.03800	26.05866	-50.92779	10.93792
3	2001	48.24750	-1.89837	36.29772	2.58776	0.97170	-26.21683	5.05097	-10.34124	3.55005	...	34.57337	-171.70734	-16.96705	-46.67617
4	2001	50.97020	42.20998	67.09964	8.46791	-15.85279	-16.81409	-12.48207	-9.37636	12.63699	...	9.92661	-55.95724	64.92712	-17.72522
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
515340	2006	51.28467	45.88068	22.19582	-5.53319	-3.61835	-16.36914	2.12652	5.18160	-8.66890	...	4.81440	-3.75991	-30.92584	26.33968
515341	2006	49.87870	37.93125	18.65987	-3.63581	-27.75665	-18.52988	7.76108	3.56109	-2.50351	...	32.38589	-32.75535	-61.05473	56.65182
515342	2006	45.12852	12.65758	-38.72018	8.80882	-29.29985	-2.28706	-18.40424	-22.28726	-4.52429	...	-18.73598	-71.15954	-123.98443	121.26989
515343	2006	44.16614	32.38368	-3.34971	-2.49165	-19.59278	-18.67098	8.78428	4.02039	-12.01230	...	67.16763	282.77624	-4.63677	144.00125
515344	2005	51.85726	59.11655	26.39436	-5.46030	-20.69012	-19.95528	-6.72771	2.29590	10.31018	...	-11.50511	-69.18291	60.58456	28.64599

515345 rows x 91 columns

Figure 1: Dataset

Since the output class is a numeric discrete value, regression can be applied to this model.
For this dataset the following algorithms are used:

- Linear Regression using sklearn
- Single Layer NN using keras
- Multilayer Layer NN using keras

5 Neural Networks

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. It is capable of complex solutions which may not be possible with simpler algorithms. It best replicates the working of a human brain on a particular machine learning problem.

A single-layer neural network represents the most simple form of neural network, in which there is only one layer of input nodes that send weighted inputs to a subsequent layer of receiving nodes, or in some cases, one receiving node. This single-layer design was part of the foundation for systems which have now become much more complex.

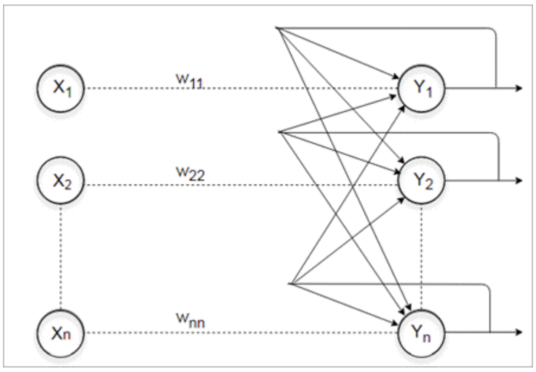


Figure 2: Example of a Single Layer Neural Network

A Multi Layer Neural Network contains one or more hidden layers (apart from one input and one output layer). While a single layer perceptron can only learn linear functions, a multi layer perceptron can also learn non – linear functions.

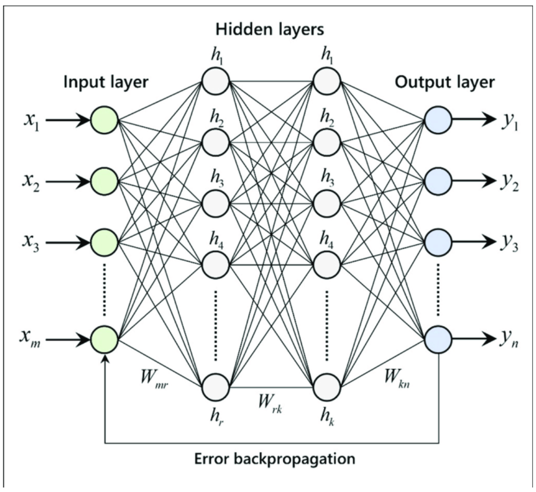


Figure 3: Example of a Multi Layer Neural Network

5.1 Preprocessing

```python
import pandas as pd
import numpy as np

df = pd.read_csv('YearPredictionMSD.txt', header=None)
df
for idx in range(X.shape[1]):
    X[:,idx]=X[:,idx]/max(X[:,idx])

train_x=X[:463715]
test_x=X[463716:]
train_y=Y[:463715]
test_y=Y[463716:]

normalizer = preprocessing.Normalization()
normalizer.adapt(np.array(train_x))

first = np.array(train_x[:1])

with np.printoptions(precision=2, suppress=True):
    print('First example:', first)
    print()
    print('Normalized:', normalizer(first).numpy())
```

```
First example: [[  49.94   21.47   73.08    8.75  -17.41  -13.1   -25.01  -12.23    7.83
    -2.47    3.32   -2.32   10.21  611.11  951.09  698.11  408.98  383.71
   326.52  238.11  251.42  187.17  100.43  179.19   -8.42 -317.87   95.86
    48.1   -95.66  -18.06    1.97   34.42   11.73    1.37    7.79   -0.37
  -133.68  -83.26  -37.3    73.05  -37.37   -3.14  -24.22  -13.23   15.94
   -18.6    82.15  240.58  -10.29   31.58  -25.38   -3.91   13.29   41.55
    -7.26  -21.01  105.51   64.3    26.08  -44.59   -8.31    7.94  -10.74
   -95.45  -82.03  -35.59    4.7    70.96   28.09    6.02  -37.14  -41.12
    -8.41    7.2    -8.6    -5.91  -12.32   14.69  -54.32   40.15   13.02
   -54.41   58.99   15.37    1.11  -23.09   68.41   -1.82  -27.46    2.26]]

Normalized: [[ 1.08  0.39  1.83  0.47 -0.48 -0.28 -1.55 -1.31  0.39 -0.67  0.79 -0.58
  -1.06 -1.04 -0.8  -0.74 -1.05 -0.86 -0.87 -0.9  -0.67 -0.83 -1.01 -0.73
  -0.42 -0.5   0.26  0.35 -0.68 -0.47 -0.03  0.15  0.03  0.1   0.17 -0.68
  -0.2  -0.44  0.58  0.24 -0.3  -0.18  0.38 -0.43  0.42 -0.46  0.01  0.37
   0.36  0.05 -0.34 -0.43  0.01  0.48  0.05 -0.31  0.    0.24 -0.08 -0.11
  -0.19  0.15 -0.27  0.14 -0.37 -0.28  0.02  0.37 -0.04  0.19 -0.11 -0.2
   0.11  0.3   0.2  -0.01  0.04 -0.12  0.25  0.11 -0.08  0.11  0.14 -0.24
   0.05 -0.36  0.54 -0.47 -0.26  0.04]]
```

Figure 4: Normalized input

5.2 Model Preparation

Here we train the neural network and calculate the loss for it using the predict function using a single layer and multilayer neural network.

5.2.1 NN with keras

```python
 linear_model = tf.keras.Sequential([
     normalizer,
     layers.Dense(units=89)
 ])

 learning_rate=0.1
 epoch=10
```

```
linear_model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=learning_rate),
    loss='mean_absolute_error')
# mean_squared_error
history = linear_model.fit(
    train_x, train_y,
    epochs=epoch,
    # suppress logging
    verbose=0,
    # Calculate validation results on 20% of the training data
    validation_split = 0.2)
import matplotlib.pyplot as plt

def plot_loss(history):
    plt.plot(history.history['loss'], label='loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.ylim([0, 10])
    plt.xlabel('Epoch')
    plt.ylabel('Error [MPG]')
    plt.legend()
    plt.grid(True)
    plot_loss(history)

linear_model.evaluate(test_x, test_y, verbose=0)
test_predictions = linear_model.predict(test_x)
error = test_predictions - test_y
# Print MSE
sum(sum(np.square(error)))/2/len(test_y)
```

5.2.2 DNN with keras

```
  learning_rate=0.1
  epoch=10
  def build_and_compile_model(norm):
      model = keras.Sequential([
        norm,
        layers.Dense(90, activation='relu'),
  #       layers.Dense(180, activation='relu'),
        layers.Dense(135, activation='relu'),
        layers.Dense(91, activation='relu'),
        layers.Dense(89)
      ])

      model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.Adam(learning_rate))
      return model
  dnn_model = build_and_compile_model(normalizer)
  dnn_model.summary()

  history = dnn_model.fit(
      train_x, train_y,
      validation_split=0.2,
      verbose=0, epochs=epoch)

  import matplotlib.pyplot as plt

  def plot_loss(history):
```

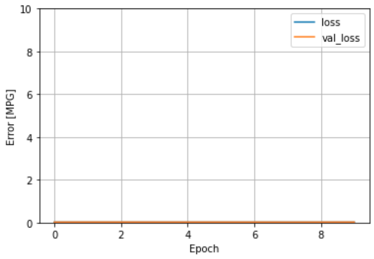
```
        plt.plot(history.history['loss'], label='loss')
        plt.plot(history.history['val_loss'], label='val_loss')
        plt.ylim([0, 10])
        plt.xlabel('Epoch')
        plt.ylabel('Error [MPG]')
        plt.legend()
        plt.grid(True)
    plot_loss(history)

    dnn_model.evaluate(test_x, test_y, verbose=0)
        test_predictions = dnn_model.predict(test_x)
        error = test_predictions - test_y
    # Print MSE
    sum(sum(np.square(error)))/2/len(test_y)

```

5.3 Model Testing

Testing the neural network, we get the mean square error as follows:

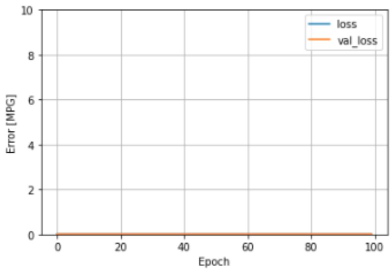
Figure 5: MSE for NN 0.1 learning rate and epoch=10

```
plot_loss(history)
```




```
In [12]: dnn_model.evaluate(test_x, test_y, verbose=0)
Out[12]: 0.024177128449082375

In [15]: test_predictions = dnn_model.predict(test_x)
         error = test_predictions - test_y
         error

Out[15]: <tf.Tensor: shape=(51629, 89), dtype=float32, numpy=
         array([[ 0.00813567,  0.00017492,  0.01474756, ..., -0.00313638,
                 -0.00378672,  0.00121196],
                [ 0.00813567,  0.00017492,  0.01474756, ..., -0.00313638,
                 -0.00378672,  0.00121196],
                [ 0.00813567,  0.00017492,  0.01474756, ..., -0.00313638,
                 -0.00378672,  0.00121196],
                ...,
                [ 0.00813567,  0.00017492,  0.01474756, ..., -0.00313638,
                 -0.00378672,  0.00121196],
                [ 0.00813567,  0.00017492,  0.01474756, ..., -0.00313638,
                 -0.00378672,  0.00121196],
                [ 0.00813567,  0.00017492,  0.01474756, ..., -0.00313638,
                 -0.00378672,  0.00121196]], dtype=float32)>

In [13]: sum(sum(np.square(error)))/2/len(test_y)

Out[13]: 0.5059558361035429
```

Figure 6: MSE for DNN 0.1 learning rate and epoch=10




```
In [18]: dnn_model.evaluate(test_x, test_y, verbose=0)
Out[18]: 0.012127500027418137

In [19]: test_predictions = dnn_model.predict(test_x)
         error = test_predictions - test_y
         error

Out[19]: <tf.Tensor: shape=(51629, 89), dtype=float32, numpy=
         array([[-2.2038014e-04,  6.0496491e-04, -2.0920914e-03, ...,
                  6.7315006e-05,  1.8412957e-03, -9.7641809e-04],
                [-2.2038014e-04,  6.0496491e-04, -2.0920914e-03, ...,
                  6.7315006e-05,  1.8412957e-03, -9.7641809e-04],
                [-2.2038014e-04,  6.0496491e-04, -2.0920914e-03, ...,
                  6.7315006e-05,  1.8412957e-03, -9.7641809e-04],
                ...,
                [-2.2038014e-04,  6.0496491e-04, -2.0920914e-03, ...,
                  6.7315006e-05,  1.8412957e-03, -9.7641809e-04],
                [-2.2038014e-04,  6.0496491e-04, -2.0920914e-03, ...,
                  6.7315006e-05,  1.8412957e-03, -9.7641809e-04],
                [-2.2038014e-04,  6.0496491e-04, -2.0920914e-03, ...,
                  6.7315006e-05,  1.8412957e-03, -9.7641809e-04]], dtype=float32)>

In [20]: sum(sum(np.square(error)))/2/len(test_y)
Out[20]: 0.49963573328613864
```

Figure 7: MSE for DNN 0.01 learning rate and epoch=100

6 Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. Hence, it tried to explain the relationship between the output Y and input(s) X by creating an equation for a line.

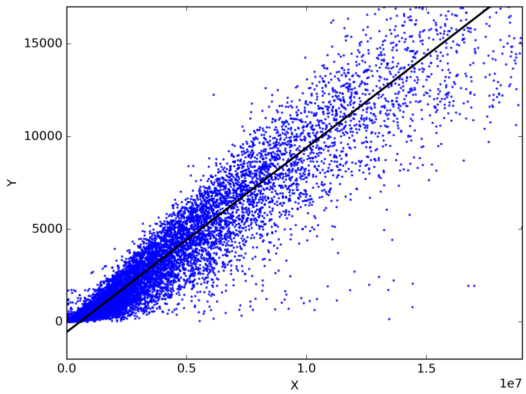


Figure 8: Example of a linear regression model fitting

6.1 Preprocessing

```
import pandas as pd
import numpy as np

df = pd.read_csv('YearPredictionMSD.txt', header=None)
df
for idx in range(X.shape[1]):
    X[:,idx]=X[:,idx]/max(X[:,idx])

train_x=X[:463715]
test_x=X[463716:]
train_y=Y[:463715]
test_y=Y[463716:]
```

6.2 Model Preparation

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(train_x, train_y)

r_sq = model.score(train_x, train_y)

print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
```

6.3 Model Testing

```
y_pred = model.predict(test_x)
y_pred=np.around(y_pred)

```

```python
print('predicted response:', y_pred, sep='\n')
print('actual response:', test_y, sep='\n')

from sklearn.metrics import mean_squared_error
mean_squared_error(test_y,y_pred)
```

```python
print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
```

```
coefficient of determination: 0.3602828223068081
intercept: 1969.1274332562098
slope: [ 31.99683631  -9.77544382  -3.40378969 -15.39830478  16.25861626
 -45.67552259  -3.19350038  -0.61672409   7.93201453  -0.35578293
 -21.45137866  22.24266597 -32.78038255   5.23110072 -29.14189796
  15.34648434  13.43595115  79.73408731 -21.87146977 102.39474909
  -9.4270343  -51.82235668 -24.46285518  -1.06584318   8.74403999
 -37.08875404   8.32614338 -16.06079312  58.73159      37.04968064
 -25.80369195 -21.28523073  18.16078946   8.13672016  28.19854803
   0.7209632  -13.47509332   3.49620344 -24.71320915  37.21186424
  23.03969494  39.21651129   2.51669537  19.29750877  63.92556694
  -5.88159329   8.92052974 -12.21537432  26.34829118 -25.00555365
  -8.79449499   2.49277095 -61.0518601   75.71623298 -25.60892549
 -30.59776697  19.46240851  -0.53951436 -12.79946087   9.97701374
   0.17772797  -6.683211     4.82130752   9.7429527  -35.36484008
 -19.1143826  -18.48040118  32.95391027  -7.88734508  16.5522544
  -5.7644389   -6.25066557 -16.73583417 -36.82316118  30.25973589
  10.3951017   -1.90816456 -36.19017985  26.99644702  -0.75357372
  -0.51186282   0.87699778  -8.00938572  -8.36472649 -13.21635156
   0.41984244]
```

```python
from sklearn.metrics import confusion_matrix, mean_squared_error
mean_squared_error(test_y,y_pred)
```

```
738.25
```

```python
len([0 for ele in test_y-y_pred if ele==0])/len(y_pred)*100
```

```
5.0
```

Figure 9: Linear Regression params and error

7 Conclusion

Comparing the mean-sqaure error term of the above alogorithms we get:

Algorithm	MSE
NN with learning rate=0.1 and epoch=10	43.427
DNN with learning rate=0.1 and epoch=10	0.5059
DNN with learning rate=0.1 and epoch=100	0.4996
Linear Regression	738.25

We see that DNN is by far a better option when selecting the algorithm, as it has the least mean square error amongst the other models.

Multilayer networks solve the classification problem for non linear sets by employing hidden layers, whose neurons are not directly connected to the output. The additional hidden layers can be interpreted geometrically as additional hyper-planes, which enhance the separation capacity of the network. Hence, we can conclude that DNN(multi layered neural networks) is more suitable for regression problems whose solution is not linear and with many features.