

Assignment 1

I used simplest ways to do this assignment. (No complex things) I used simple linear regression, Polynomial regression, Ridge (l2) with different alphas.

Question:

You are given 1000 images of size 28x28 pixels. Each image contains a single line drawn at a specific angle, and the target for each image is that specific angle. The image pixels are flattened row-wise, resulting in 784-pixel values per image. The target angle is appended as the 785th entry. Therefore, the dataset has 1000 rows and 785 columns. Your task is to develop a regression model to predict the angle of the line from the image data. The input will be the flattened image data (784-pixel values), and the output will be the predicted angle.

1. Loading The data

```
import pandas as p

traindata=p.read_csv("train.csv")
testdata=p.read_csv("test.csv")
testdata.drop(columns=['id'],inplace=True)

traindata.head(25)
```

	pixel_0	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	pixel_9	...	pixel_775	pixel_776	pixel_777	pixel_778	pixel_779	pixel_780	pixel_781	pixel_782	pixel_783
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

2. Feature extraction

```
x=traindata.drop(columns=['angle'])
y=traindata['angle']
```

```
print(x.shape)
print(y.shape)
```

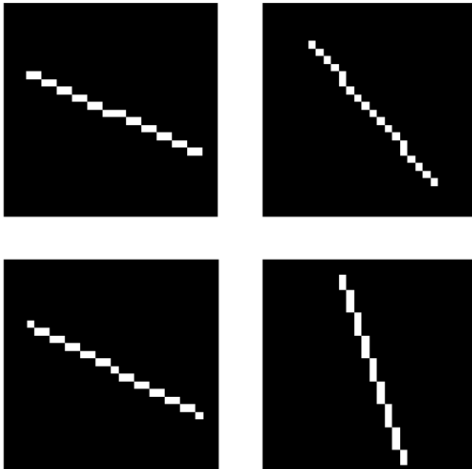
```
(1000, 784)
(1000,)
```

3. Plotting the pixels:

```
[ ] import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2, figsize=(6, 6))

for i, ax in enumerate(axes.flat):
    ax.imshow(x.iloc[i].values.reshape(28, 28), cmap='gray')
    ax.axis('off')
plt.show()
```



4. Splitting

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

5. StandardScaler

```
] from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
x_train_scaled = scaler.fit_transform(x_train, y_train)
x_test_scaled = scaler.transform(x_test)
testdatascaled=scaler.transform(testdata)
```

The pixels intensity value (RGB) differs from 0 to 255, so as to scale them I used StandardScaler. I think If I use MinMaxScaler it may perform even well. I don't know, but for StandardScaler according to formula, when it calculates mean of the data, here in this data many columns mean will be nearer to zero or zero, and many of them will way far to zero. Using MinMax may be beneficial here.

6. PCA

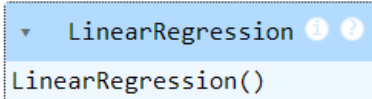
```
from sklearn.decomposition import PCA

pca = PCA(n_components=150)
xtrainpcaed=pca.fit_transform(x_train_scaled)
xtestpcaed=pca.transform(x_test_scaled)
testdatapcaed=pca.transform(testdatascaled)
```

As the data has 784 dimensions, I used PCA. I started with PCA(n_components=2) MSE was high so I started increasing the value of n_components.

7. Linear Regression

```
from sklearn.linear_model import LinearRegression  
  
lr=LinearRegression()  
lr.fit(xtrainpcaed,y_train)
```



A screenshot of a Jupyter Notebook cell. It shows a dropdown menu for the 'LinearRegression' class, with the option 'LinearRegression()' selected. The cell also contains the code 'LinearRegression()'.

After doing StandardScaler and PCA, I trained simple linear regression model with xtrainpcaed, y_train.

8. Prediction

```
y_pred=lr.predict(xtestpcaed)  
  
# ids = range(1, len(y_pred) + 1)  
# submission_df = pd.DataFrame({'id': ids, 'angle': y_pred})  
# submission_df.to_csv('submission.csv', index=False)  
  
from sklearn.metrics import mean_squared_error  
mean_squared_error(y_test, y_pred)  
  
1.8157296253959603
```

Then predicted with xtestpcaed, and found MSE = 1.815729..., each time I ran this linear regression part the MSE was varying. The least MSE for linear reg. achieved was 1.77911 for testdatapcaed not xtestpcaed, which can be found in my Kaggle submissions.

9. Polynomial reg

```
from sklearn.preprocessing import PolynomialFeatures  
  
poly = PolynomialFeatures(degree=2)  
xtrainpcaed=poly.fit_transform(xtrainpcaed)  
  
lr.fit(xtrainpcaed,y_train)  
  
from sklearn.metrics import mean_squared_error  
testdatapcaed=poly.transform(testdatapcaed)  
y_pred=lr.predict(testdatapcaed)
```

Then I applied polynomial reg with degree = 2, and fitted the model, and predicted for testdatapcaed, as the testdatapcaed has dimensions [500,150] and the xtrainpcaed data had [800,150], MSE can't be calculated, so I directly uploaded it under Kaggle competition, it gave RMSE = 1.74674.

10. Ridge

```
from sklearn.linear_model import Ridge

ridge_model = Ridge(alpha=1)
ridge_model.fit(xtrainpcaed, y_train)
y_pred = ridge_model.predict(testdatapcaed)
```

To make the model to be less overfitting I added $\lambda(m^2)$ term. From the library `sklearn.linear_model` in class `Ridge`, when I started increasing the parameter `alpha` starting from 1 to 10, the MSE started decreasing for the same. And it was like – 1.74304, 1.74295, 1.74286, 1.74277, 1.74261 if I again Increase the `alpha` value the MSE value will be more less. But it will be underfitting model. It will be good for `xtrainpcaed`, not for `testdatapcaed`.