0. Kuo Zhang

1. There was problem during class.
Why GD Gradient descent objective value always decreasing? While under proper conditions?

First we have to make an assumption function

$\forall t, F(w^{t+1}) - F(w^t) < 0 \Leftrightarrow F(w^{t+1}) \le F(w^t)$

In this function $F(w^{t+1})$ is learn rate $F(w^t)$ is current.
Then we have to proof this function to proof this assumption is right:

$w^{t+1} = w^t - \eta \cdot \nabla F(w^t)$

$\cancel{F(w^{t+1}) - F(w^t) = \langle \nabla F(w), w^{t+1} - w^t \rangle}$

$\Uparrow$

~~Apply Mean Value theorem:~~

$F: R \to R \longrightarrow F: R^d \to R$

$\cancel{\forall x, y \in R}$ $\qquad \forall x, y \in R^d$

$F(x) - F(y) = F'(z) \cdot (x - y)$ $\qquad F(x) - F(y) = \langle \nabla F(z), x - y \rangle$

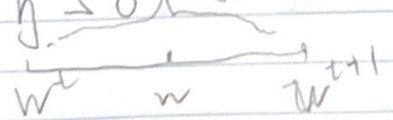
$z \in [x, y]$

~~Now, we have Function:~~

$$F(w^{t+1}) - F(w^t) = \langle \nabla F(w), w^{t+1} - w^t \rangle$$
$$= \langle \nabla F(w), -\eta \cdot \nabla F(w^t) \rangle$$
$$= -\eta \cdot \langle \nabla F(w), \nabla F(w^t) \rangle$$

So we know that,

$\eta \to 0 \wedge \approx 0$



If line segement goes to zero size, w must be very close to w^t

So $\nabla F(w) \approx \nabla F(w^t)$
It's in same vector is positive but, overall is negative, $-\eta$.
Hence, GD Gradient descent always decrease objective function

2. Random projection and principal component analysis are two methods that can be used to reduce the dimentionality of a dataset. Both method are useful for making data easier to work with but they are different in effective at different type of data

1) The random project would outperform PCA when analyzing a large-scale dataset, while PCA would outperform the random projection when the dimension and sample size relatively small. This is because that the random projection has the time complexity of $O(npd)$
And Complexity of PCA is $O(np^2 + p^3)$

2) In mixture of Gaussians, random projection are fast reducing the dimension

3). PCA have to hold data in memory for large data but Random project don't need to do so.

4). Random projection will not getting worst after initial training data But, PCA will provide best projection and it will gone worse after that.

Conclusion:
Random projection projecting data at lower dimensional space via randomly generated matrix, good for reducing high dimension, but will lose accuracy and information.

PCA is accurate for dimension reduction but it can be computationally intensive of large data set

3. Example active learning lead to exponetial saving:

For active learning, we can choose to use a crowdsourcing-like data set but do not have to label the entire data set. The active learning algorithm iteratively selects data based on some metric and sends this unlabeled data to the authority which then labels it and returns it to algorithm.
For example:
the MNIST experiments (train_num = 55000, val_num = 10000) using the AlexNet model Pytorch framework: 1) using the full training data to directly train the model 120 times epoch, val_acc = 98.992%
2) Using Uncertainty Strategy
only 2300 labeled data val_acc needed to reach 99.04% and the remaining 52700 and into trained model for prediction and 99.70% (52543 / 52700) is obtained.
It can be seen that for the MNIST data set significant results can be obtained using active learning.

~~Example when active learning~~ does not:

Data selected using active learning may have a large number of duplicates, thus causing annotation redundancy problems. Typically, the information content of unlabeled points is evaluated separately. In active learning we use BALD a collection function method to evaluated unlabeled content. But since individual information points may be almost identical evaluating information at each point separetely is wasting resources.
For example if we simply acquire the top K most useful points, we may end up with having experts label K almost identical points.

4.

algorithm

1) We can use KNN. To be specific for each observation with missing value, we should compute the distances between this observation and all others without missing value based on the features except Blood Pressure. Then find the K observations corresponding to the smallest K distances and calculate the average value of their Blood Pressure. Finally use this value to fill '?'.

Let the indices corresponding to missing values be $i_1, \dots, i_N$

Algorithm:

Choose the value of K.

For $i = i_1, \dots, i_N$

① Compute $d_j = \| X_i - X_j \|_2$ for $j \rightarrow$
$j \in \{1, \dots n\} \setminus \{i_1, \dots i_N\}$.
where X_i is the vector of features without Blood pressure for the ith observation

② Find the K indices corresponding to the smallest K distances: $j_1, \dots j_k$

③ Fill the missing value by $1/k \sum_{i=1}^{k} y_{j_i}$
where y_j is the value of Blood Pressure for jth observation

4.

2) If we know which observed entries are corrupted we can just remove them and fill the table using the algorithm from question 1).

However, if we do not know which observed entries are corrupted, we should first find them,
Use the following procedure: T: corrupted data
1. compute empirical mean and covariance μ_T, Σ_T
Compute largest eigenvalue λ^* of $\Sigma_T - I$ and, eigenvector v^*
If λ^* is small, return μ_T
Otherwise, find $t > C_1$ such that

$$\Pr_{X \in T}\{|v^* \cdot (X - \mu_T)| > t\} > C_2 e^{-t^2/2} + \frac{C_3 \varepsilon}{t^2 \log(n \log \frac{n}{\varepsilon \tau})}$$

then Remove X such that $|v^* \cdot (X - \mu_T)| > t$
return to first step.

5. ① A random forest regression model is built to predict each missing value using several unmissing values. First, let us convert the categorical variables into classification codes that can be handled by the random forest model, using first Age as an example

```
df['Age_category'] = df['Age'].astype('category')
df['Age_category'] = df['Age_category'].cat.codes
```

② Increasing the random sample size:

```
df = pd.read_csv("xxx.csv").sample(n=50, random_state=42)
```

③ Import the random forest regressor module from Scikit learn. Define the list of features used to train the model:

```
from sklearn.ensemble import RandomForestRegressor
features = ['Age_category', 'Weight_category', 'Height_category' ...... ]
```

④ Train our model with a random forest 1000 estimates and a maximum depth of 1000. then generate the predictions and add to new list

```
for train_index, test_index in kd.split(df_filter):
    df_test = df_filter.iloc[test_index]
    df_train = df_filter.iloc[train_index]
    X_train = np.array(df_train[features])
    y_train = np.array(df_train['missing value'])
    X_test = np.array(df_test[features])
    y_test = np.array(df_test['missing_value'])
```

5.-1

```
model = Random Forest Regressor (n_estimators = 1000,
max-depth = 1000, random - state = 42)
model.fit (X-train, y-train)
y-pred-rf append (model.predict (X-test)[0]).
y-true-rf.append (y-test[0])
```

⑤ mean-squred-error (y-pred-rf, y-true-rf).

⑥ Predicting missing values

```
model_rf = Random Forest Regressor (n_estimators =
1000, max-depth = 1000, random-state = 42)
model_rf.fit (X-train-rf, y-train-rf)
mode_rf.predict (X-test_rf)[0])
```