1. Image dataset

Model Description (algorithms)

- Using 3 kinds of CNN models to compare with each other:

A. Normal CNN model.

B. CNN model with augmented data by rotating, shifting and zooming in the original data to generate new data.

C. CNN model trained with less amount of data.

- Both constructed by the same layers below, with 64 batch size and 20 epochs.

```
model1 = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Conv2D(128, (3, 3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(32, activation=tf.nn.relu),
        tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
```

Dataset Description

- [Intel image Classification dataset from Kaggle]

The goal is to tell whether a picture is about mountain, building, street etc.

Training Process

Model A:

```
Epoch 1/20
44/44 [==============================] - 15s 78ms/step - loss: 1.7346 - accuracy: 0.3609 - val_loss: 1.2418 - val_accuracy: 0.4754
Epoch 2/20
44/44 [==============================] - 2s 50ms/step - loss: 1.1500 - accuracy: 0.5483 - val_loss: 1.1101 - val_accuracy: 0.5303
Epoch 3/20
44/44 [==============================] - 2s 53ms/step - loss: 0.9377 - accuracy: 0.6340 - val_loss: 0.9591 - val_accuracy: 0.6185
Epoch 4/20
44/44 [==============================] - 2s 54ms/step - loss: 0.7875 - accuracy: 0.7074 - val_loss: 0.9405 - val_accuracy: 0.6561
Epoch 5/20
44/44 [==============================] - 3s 59ms/step - loss: 0.6280 - accuracy: 0.7736 - val_loss: 0.9326 - val_accuracy: 0.6633
Epoch 6/20
44/44 [==============================] - 2s 55ms/step - loss: 0.4431 - accuracy: 0.8557 - val_loss: 0.9054 - val_accuracy: 0.6590
Epoch 7/20
44/44 [==============================] - 2s 51ms/step - loss: 0.3346 - accuracy: 0.8937 - val_loss: 1.0070 - val_accuracy: 0.6618
Epoch 8/20
44/44 [==============================] - 2s 53ms/step - loss: 0.2220 - accuracy: 0.9331 - val_loss: 1.1373 - val_accuracy: 0.6373
Epoch 9/20
44/44 [==============================] - 2s 53ms/step - loss: 0.1436 - accuracy: 0.9624 - val_loss: 1.0949 - val_accuracy: 0.6864
Epoch 10/20
44/44 [==============================] - 2s 51ms/step - loss: 0.0834 - accuracy: 0.9812 - val_loss: 1.1681 - val_accuracy: 0.6936
Epoch 11/20
44/44 [==============================] - 3s 59ms/step - loss: 0.0638 - accuracy: 0.9826 - val_loss: 1.2232 - val_accuracy: 0.6850
Epoch 12/20
44/44 [==============================] - 2s 51ms/step - loss: 0.0362 - accuracy: 0.9946 - val_loss: 1.4648 - val_accuracy: 0.6590
Epoch 13/20
44/44 [==============================] - 2s 54ms/step - loss: 0.0217 - accuracy: 0.9975 - val_loss: 1.3843 - val_accuracy: 0.6835
Epoch 14/20
44/44 [==============================] - 2s 51ms/step - loss: 0.0123 - accuracy: 0.9989 - val_loss: 1.4705 - val_accuracy: 0.6994
Epoch 15/20
44/44 [==============================] - 2s 51ms/step - loss: 0.0109 - accuracy: 0.9989 - val_loss: 1.4350 - val_accuracy: 0.6922
Epoch 16/20
44/44 [==============================] - 2s 56ms/step - loss: 0.0086 - accuracy: 0.9996 - val_loss: 1.5742 - val_accuracy: 0.6936
Epoch 17/20
44/44 [==============================] - 2s 56ms/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 1.6450 - val_accuracy: 0.6994
Epoch 18/20
44/44 [==============================] - 2s 51ms/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 1.6620 - val_accuracy: 0.6980
Epoch 19/20
44/44 [==============================] - 2s 51ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 1.7023 - val_accuracy: 0.7110
Epoch 20/20
44/44 [==============================] - 2s 56ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 1.7648 - val_accuracy: 0.7009
```

Model B:

```
Epoch 1/20
55/55 [==============================] - 21s 353ms/step - loss: 1.7973 - accuracy: 0.3839
Epoch 2/20
55/55 [==============================] - 20s 360ms/step - loss: 1.1676 - accuracy: 0.5375
Epoch 3/20
55/55 [==============================] - 20s 366ms/step - loss: 1.0473 - accuracy: 0.5918
Epoch 4/20
55/55 [==============================] - 20s 357ms/step - loss: 1.0192 - accuracy: 0.6014
Epoch 5/20
55/55 [==============================] - 21s 391ms/step - loss: 1.2162 - accuracy: 0.4984
Epoch 6/20
55/55 [==============================] - 19s 342ms/step - loss: 0.9971 - accuracy: 0.6130
Epoch 7/20
55/55 [==============================] - 20s 357ms/step - loss: 0.9709 - accuracy: 0.6378
Epoch 8/20
55/55 [==============================] - 20s 363ms/step - loss: 0.9209 - accuracy: 0.6566
Epoch 9/20
55/55 [==============================] - 19s 343ms/step - loss: 0.9296 - accuracy: 0.6509
Epoch 10/20
55/55 [==============================] - 21s 380ms/step - loss: 0.8457 - accuracy: 0.6885
Epoch 11/20
55/55 [==============================] - 19s 341ms/step - loss: 0.8540 - accuracy: 0.6789
Epoch 12/20
55/55 [==============================] - 21s 382ms/step - loss: 0.8279 - accuracy: 0.6968
Epoch 13/20
55/55 [==============================] - 19s 339ms/step - loss: 0.7835 - accuracy: 0.7191
Epoch 14/20
55/55 [==============================] - 21s 384ms/step - loss: 0.7722 - accuracy: 0.7278
Epoch 15/20
55/55 [==============================] - 19s 347ms/step - loss: 0.7336 - accuracy: 0.7301
Epoch 16/20
55/55 [==============================] - 29s 530ms/step - loss: 0.7229 - accuracy: 0.7440
Epoch 17/20
55/55 [==============================] - 20s 367ms/step - loss: 0.8824 - accuracy: 0.6821
Epoch 18/20
55/55 [==============================] - 19s 343ms/step - loss: 0.7488 - accuracy: 0.7287
Epoch 19/20
55/55 [==============================] - 21s 382ms/step - loss: 0.7349 - accuracy: 0.7368
Epoch 20/20
55/55 [==============================] - 19s 343ms/step - loss: 0.6730 - accuracy: 0.7588
```

Model C:

```
Epoch 1/20
22/22 [==============================] - 5s 138ms/step - loss: 2.3301 - accuracy: 0.2337 - val_loss: 1.4425 - val_accuracy: 0.3526
Epoch 2/20
22/22 [==============================] - 1s 53ms/step - loss: 1.3771 - accuracy: 0.4291 - val_loss: 1.2853 - val_accuracy: 0.5347
Epoch 3/20
22/22 [==============================] - 1s 52ms/step - loss: 1.1962 - accuracy: 0.5535 - val_loss: 1.1318 - val_accuracy: 0.5549
Epoch 4/20
22/22 [==============================] - 1s 54ms/step - loss: 0.9626 - accuracy: 0.6346 - val_loss: 1.1380 - val_accuracy: 0.5780
Epoch 5/20
22/22 [==============================] - 1s 51ms/step - loss: 0.7129 - accuracy: 0.7344 - val_loss: 0.9408 - val_accuracy: 0.6387
Epoch 6/20
22/22 [==============================] - 1s 51ms/step - loss: 0.5181 - accuracy: 0.8155 - val_loss: 1.1723 - val_accuracy: 0.6012
Epoch 7/20
22/22 [==============================] - 1s 54ms/step - loss: 0.4108 - accuracy: 0.8719 - val_loss: 1.1192 - val_accuracy: 0.6387
Epoch 8/20
22/22 [==============================] - 1s 54ms/step - loss: 0.2969 - accuracy: 0.9009 - val_loss: 1.0089 - val_accuracy: 0.6387
Epoch 9/20
22/22 [==============================] - 1s 58ms/step - loss: 0.1982 - accuracy: 0.9508 - val_loss: 1.0568 - val_accuracy: 0.6618
Epoch 10/20
22/22 [==============================] - 1s 56ms/step - loss: 0.1435 - accuracy: 0.9616 - val_loss: 1.2782 - val_accuracy: 0.6590
Epoch 11/20
22/22 [==============================] - 1s 54ms/step - loss: 0.1198 - accuracy: 0.9740 - val_loss: 1.1707 - val_accuracy: 0.6676
Epoch 12/20
22/22 [==============================] - 1s 54ms/step - loss: 0.0805 - accuracy: 0.9855 - val_loss: 1.1839 - val_accuracy: 0.6445
Epoch 13/20
22/22 [==============================] - 1s 52ms/step - loss: 0.0903 - accuracy: 0.9696 - val_loss: 1.2370 - val_accuracy: 0.6445
Epoch 14/20
22/22 [==============================] - 1s 54ms/step - loss: 0.1132 - accuracy: 0.9725 - val_loss: 1.5946 - val_accuracy: 0.5665
Epoch 15/20
22/22 [==============================] - 1s 54ms/step - loss: 0.0499 - accuracy: 0.9920 - val_loss: 1.3074 - val_accuracy: 0.6503
Epoch 16/20
22/22 [==============================] - 1s 52ms/step - loss: 0.0212 - accuracy: 0.9986 - val_loss: 1.3096 - val_accuracy: 0.6474
Epoch 17/20
22/22 [==============================] - 1s 52ms/step - loss: 0.0155 - accuracy: 0.9986 - val_loss: 1.4694 - val_accuracy: 0.6387
Epoch 18/20
22/22 [==============================] - 1s 54ms/step - loss: 0.0095 - accuracy: 0.9993 - val_loss: 1.4220 - val_accuracy: 0.6387
Epoch 19/20
22/22 [==============================] - 1s 55ms/step - loss: 0.0066 - accuracy: 1.0000 - val_loss: 1.4637 - val_accuracy: 0.6590
Epoch 20/20
22/22 [==============================] - 1s 62ms/step - loss: 0.0051 - accuracy: 1.0000 - val_loss: 1.4907 - val_accuracy: 0.6445
```

Performances

Model A:

	precision	recall	f1-score	support
buildings	0.68	0.57	0.62	437
forest	0.86	0.88	0.87	474
glacier	0.70	0.67	0.69	553
mountain	0.64	0.69	0.66	525
sea	0.69	0.67	0.68	510
street	0.71	0.78	0.74	501
micro avg	0.71	0.71	0.71	3000
macro avg	0.71	0.71	0.71	3000
weighted avg	0.71	0.71	0.71	3000
samples avg	0.71	0.71	0.71	3000

Model B:

	precision	recall	f1-score	support
buildings	0.57	0.74	0.64	437
forest	0.78	0.97	0.86	474
glacier	0.79	0.75	0.77	553
mountain	0.74	0.75	0.74	525
sea	0.89	0.44	0.59	510
street	0.75	0.79	0.77	501
micro avg	0.74	0.74	0.74	3000
macro avg	0.75	0.74	0.73	3000
weighted avg	0.76	0.74	0.73	3000
samples avg	0.74	0.74	0.74	3000

Model C:

	precision	recall	f1-score	support
buildings	0.60	0.56	0.58	437
forest	0.77	0.87	0.82	474
glacier	0.67	0.57	0.61	553
mountain	0.58	0.69	0.63	525
sea	0.61	0.56	0.59	510
street	0.70	0.70	0.70	501
micro avg	0.66	0.66	0.66	3000
macro avg	0.65	0.66	0.65	3000
weighted avg	0.65	0.66	0.65	3000
samples avg	0.66	0.66	0.66	3000

Analysis

- We can discover from Model A and C that When using different amounts of training data, the one with more data has a more complete training process, which leads to a more comprehensive model with better performance.
- We can discover from Model A and B that when using augmentation, it is equivalent to having more data in the dataset, which makes the model learn similar data to boost the performance of similar images.

2. Non-Image dataset

Model Description (algorithms)

- Using RandomForest and Adaboost with ExtraTreeClassifier:
  A. RandomForest: n_estimators=3 and max_depth=100 (best result after adjusting the hyperparameters.)
  B. Adaboost using Extratree Classifier.

Dataset Description

- Loan prediction dataset from Univ.AI hackathon contest
  The goal is to determine whether the person has risk of unable to return the loan given some personal information including age, job, married or not, etc.

Data Preprocessing

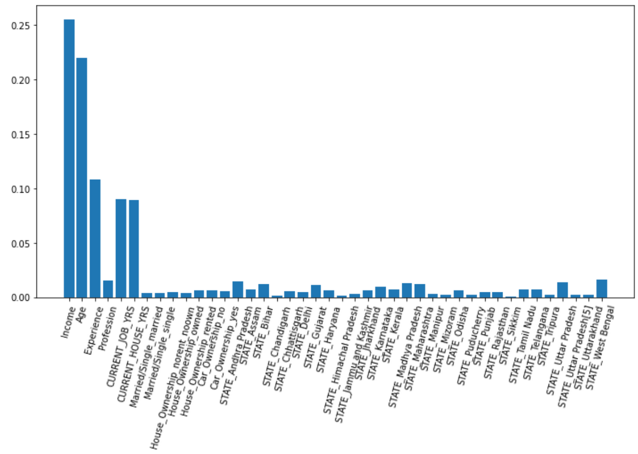
- As the dataset has some features with high cardinality (also related with others), I use feature hashing (a kind of MLP technique, which translate each word into a different token by one-hot encoding).
- Some features (such as living state) are quite hard to do one-hot encoding or feature hashing, so just drop it

from the dataset.

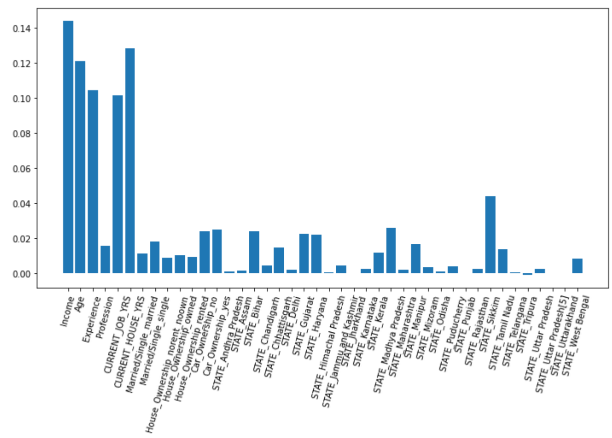
Training Process

- As both classifiers are decision tree based models, cross entropy is used to tell the purity of each node and then separate further in the deeper ones. Finally, we will get feature importance from the model to measure how critical each feature is.

Model A:



Model B:

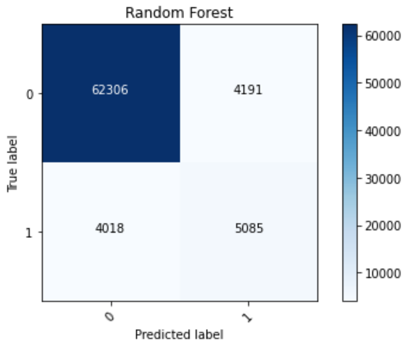


Performances

- Performance is measured by accuracy and confusion matrix.

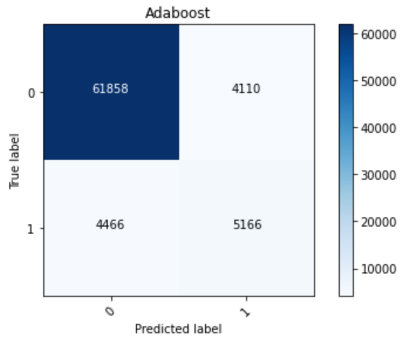
  <u>Model A:</u>

  accuracy: 0.8914153439153439



  <u>Model B:</u>

  accuracy: 0.8865608465608465



Analysis

- We can discover that although these two are both decision tree based classifiers, the resulting model trained through the process are slightly different. As extratree classifier uses the same dataset to train every tree while random forest uses possibly repeated partial dataset, the importance of some features might be higher as the feature is selected more times.

3. Self-made dataset

Model Description (algorithms)

- Using KNN with 2 different hyper parameters:

  A. N_neighbors = 3

  B. N_neighbors = 5

Dataset Description

- Using part of data collected from CWB(中央氣象局) and label them by myself. I collected information about avg, highest and lowest temperature, avg, min humidity every day from 2022/10 to 2023/2. The goal is to predict whether the day will rain or not. So I also search the precipitation for each day to label them.

Data Preprocessing

- data such as date is not numeric, here I decide to drop it. (However it might actually affect whether it is a raining day or not).

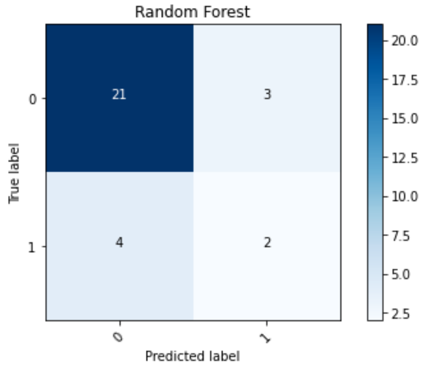
- Some data in some day is missing, here I decide to drop the day if there is partial data missing.

Performances

- Performance is measured by accuracy, precision, recall, f1 score and confusion matrix.

Model A:

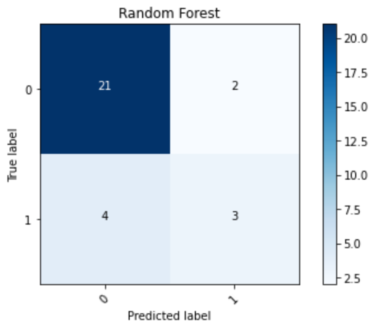
accuracy: 0.7666666666666667



	precision	recall	f1-score	support
0	0.88	0.84	0.86	25
1	0.33	0.40	0.36	5
accuracy			0.77	30
macro avg	0.60	0.62	0.61	30
weighted avg	0.78	0.77	0.77	30

Model B:

accuracy: 0.8



	precision	recall	f1-score	support
0	0.91	0.84	0.87	25
1	0.43	0.60	0.50	5
accuracy			0.80	30
macro avg	0.67	0.72	0.69	30
weighted avg	0.83	0.80	0.81	30

Analysis

- We can discover that although these two are the same classifier; however, hyper parameters also account for the result. We can observe that in whatever metrics, model B has better performance than model A, this is because when there is more n_estimators, there are more data for the classifying data to take reference.

4. Final discussion

- All the above experiments are reasonable and meet my expectations.
- We can find from the above experiments that there are several factors affecting the result, such as different hyper parameters, dataset size, data augmentation, even way to preprocess data and using different classifiers (algorithm), such as discontinuous data are not suitable for linear regression.
- If there's more time, I will use average value to fill in NA values in my self-made dataset grouped by month and preprocess the date column into month and day. Also, take data from the last day as a reference to forecast the weather the next day.
- I've learnt to use different algorithms to deal with different dataset as well as how to preprocess the data in a proper way will affect the performance immensely.

Appendix

A. image dataset

```
import numpy as np
import os
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
import tensorflow.keras.metrics as metrics
from sklearn.utils import shuffle
from tensorflow.keras.utils import to_categorical
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder

# dataset 1:  image dataset

img_datasets = ['./seg_train/seg_train', './seg_test/seg_
test']
image_size = (150, 150)


# load image from both training and testing datasets

def load_image():
  ret = []
  for dataset in img_datasets:
    images = []
    labels = []

    for folder in os.listdir(dataset):
      folder_path = os.path.join(dataset, folder)
      for file in tqdm(os.listdir(folder_path)):
        file_path = os.path.join(folder_path, file)

        image = cv2.imread(file_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  #
initially BGR from imread
```

```
        image = cv2.resize(image, image_size)

        images.append(image)
        labels.append(folder)
    ret.append((images, labels))
  return ret


(x_train, y_train), (x_test, y_test) = load_image()

# normalize feature vector outside the function can reduc
e depletion of RAM
x_train = np.array(x_train, dtype='float') / 255
x_test = np.array(x_test, dtype='float') / 255


x_train, y_train = shuffle(x_train, y_train, random_state
=25)


# do one-hot encoding on labels
encoder = LabelEncoder()
y_train = to_categorical(encoder.fit_transform(y_train))
y_test = to_categorical(encoder.fit_transform(y_test))


# model1: CNN

model1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu
', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation = 'rel
u'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
```

```
])

model1.compile(optimizer = 'adam', loss = 'categorical_cr
ossentropy', metrics=['accuracy'])
model1.fit(x_train, y_train, batch_size=64, epochs=20, va
lidation_split=0.2)

# classify by selecting the most weighted feature

from sklearn.metrics import classification_report

y_pred = model1.predict(x_test)
for row in y_pred:
  max = 0
  idx = 0
  for i, val in enumerate(row):
    if val>max:
      idx = i
      max = val
  for i in range(len(row)):
    if i!=idx:
      row[i] = 0
    else:
      row[i] = 1


classification_report(y_test, y_pred, target_names=['buil
dings', 'forest', 'glacier', 'mountain', 'sea', 'street']
)


# model2: CNN with data augmentation
from keras.preprocessing.image import ImageDataGenerator

train_datagen=ImageDataGenerator(rotation_range=15 ,
                width_shift_range=0.2 ,
                height_shift_range=0.2 ,
                shear_range=0.2 ,
```

```
                    zoom_range=0.2 ,
                    data_format='channels_last')


model2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu
', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation = 'rel
u'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])

model2.compile(optimizer = 'adam', loss = 'categorical_cr
ossentropy', metrics=['accuracy'])
model2.fit(train_datagen.flow(x_train, y_train, batch_siz
e=64), epochs=20, validation_steps=0.2)


from sklearn.metrics import classification_report
y_pred2 = model2.predict(x_test)
for row in y_pred2:
  max = 0
  idx = 0
  for i, val in enumerate(row):
    if val>max:
      idx = i
      max = val
  for i in range(len(row)):
    if i!=idx:
      row[i] = 0
    else:
      row[i] = 1
```

```
classification_report(y_test, y_pred2, target_names=['bui
ldings', 'forest', 'glacier', 'mountain', 'sea', 'street'
])


# model 3: CNN with smaller dataset

import math

model3 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu
', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation = 'rel
u'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])

x_train_less = x_train[:math.floor(len(x_train)/2)]
y_train_less = y_train[:math.floor(len(y_train)/2)]

model3.compile(optimizer = 'adam', loss = 'categorical_cr
ossentropy', metrics=['accuracy'])
model3.fit(x_train_less, y_train_less, batch_size=64, epo
chs=20, validation_split=0.2)


from sklearn.metrics import classification_report
y_pred3 = model3.predict(x_test)
for row in y_pred3:
  max = 0
  idx = 0
  for i, val in enumerate(row):
    if val>max:
      idx = i
```

```
        max = val
    for i in range(len(row)):
      if i!=idx:
        row[i] = 0
      else:
        row[i] = 1


classification_report(y_test, y_pred3, target_names=['bui
ldings', 'forest', 'glacier', 'mountain', 'sea', 'street'
])
```

B. Non-image dataset

```
import re
from itertools import islice
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import joblib
from sklearn.feature_extraction import FeatureHasher
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_curve
import itertools

# read files
train = pd.read_csv("Training Data.csv")
test = pd.read_csv("Test Data.csv")

# drop columns
train.drop(['CITY', 'Id'], inplace=True, axis=1)
test.drop(['CITY', 'ID'], inplace=True, axis=1)

# find existing patterns in the dataset
pattern = {"others": 0}
for row in train['Profession']:
```

```python
    arr = re.split('_', row)
    for e in arr:
      pattern[e] = 0

# do one-hot encoding
train = pd.get_dummies(train, columns=['Married/Single',
'House_Ownership', 'Car_Ownership', 'STATE'])
test = pd.get_dummies(test, columns=['Married/Single', 'H
ouse_Ownership', 'Car_Ownership', 'STATE'])
# feature hashing
def FH(dataset, pattern):
  # calculate vector of all rows(train)
  occurrence = []

  for row in dataset['Profession']:
    arr = re.split('_', row)
    cur_pattern = pattern.copy()
    for e in arr:
      if e in cur_pattern:
        cur_pattern[e] = cur_pattern[e] + 1
      else:
        cur_pattern['others'] = cur_pattern['others'] + 1
    occurrence.append(cur_pattern)

  # do feature hashing
  h = FeatureHasher(len(pattern))
  return h.fit_transform(occurrence).toarray()
train['Profession'] = FH(train, pattern)
test['Profession'] = FH(test, pattern)

x_train = train.loc[:, train.columns != 'Risk_Flag'].to_n
umpy()
y_train = train['Risk_Flag'].to_numpy()
x_test = test.to_numpy()

X_train, X_test, Y_train, Y_test = train_test_split(x_tra
in, y_train, test_size=0.3, random_state=1234)
```

```
# train models
clf = RandomForestClassifier(n_estimators=3, max_depth=10
0)
clf.fit(X_train, Y_train)


Y_pred = clf.predict(X_test)

# accuracy
accuracy = clf.score(X_test, Y_test)
print('accuracy:', accuracy)

# confusion matrix

cm = confusion_matrix(Y_pred, Y_test)
classes = [0, 1]
plt.figure()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues
)
plt.title('Random Forest')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
thresh = cm.max() / 2
for i, j in itertools.product(range(cm.shape[0]), range(c
m.shape[1])):
    plt.text(j, i, cm[i, j], horizontalalignment="center"
, color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# feature importance
importances = clf.feature_importances_
feature = test.columns.to_numpy()
```

```
plt.figure(figsize=(12,6))
plt.bar(feature,importances,width = 0.8)
plt.xticks(rotation = 75)


# model 2: adaboost + extratree

model2 = AdaBoostClassifier(ExtraTreeClassifier())
model2.fit(X_train, Y_train)
Y_pred2 = model2.predict(X_test)
accuracy = model2.score(X_test, Y_test)
print('accuracy:', accuracy)

# confusion matrix

cm = confusion_matrix(Y_pred2, Y_test)
classes = [0, 1]
plt.figure()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues
)
plt.title('Adaboost')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
thresh = cm.max() / 2
for i, j in itertools.product(range(cm.shape[0]), range(c
m.shape[1])):
    plt.text(j, i, cm[i, j], horizontalalignment="center"
, color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# feature importance
```

```
importance = model2.feature_importances_
feature = test.columns.to_numpy()

plt.figure(figsize=(12,6))
plt.bar(feature,importance,width = 0.8)
plt.xticks(rotation = 75)
plt.show()


# train models
clf2 = RandomForestClassifier(n_estimators=3, max_depth=1
0)
clf2.fit(X_train, Y_train)


Y_pred3 = clf2.predict(X_test)

# accuracy
accuracy = clf2.score(X_test, Y_test)
print('accuracy:', accuracy)

# confusion matrix

cm = confusion_matrix(Y_pred3, Y_test)
classes = [0, 1]
plt.figure()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues
)
plt.title('Random Forest')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
thresh = cm.max() / 2
for i, j in itertools.product(range(cm.shape[0]), range(c
m.shape[1])):
```

```
    plt.text(j, i, cm[i, j], horizontalalignment="center"
, color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# feature importance
importances = clf.feature_importances_
feature = test.columns.to_numpy()

plt.figure(figsize=(12,6))
plt.bar(feature,importances,width = 0.8)
plt.xticks(rotation = 75)
```

Self-made dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classificat
ion_report

def isnum(num):
  try:
      float(num)
      return True
  except ValueError:
      return False

df = pd.read_csv('self-made.csv')

# data preprocessing by discarding NA values and convert
all values to float
df.drop(columns=['precipitation', 'date'], inplace=True)
```

```
for i, row in enumerate(df.iloc()):
  for each in row:
    if not (isnum(each)):
      df = df.drop(i)
      break
df.astype(float)

x = np.array(df.drop(['rain'], axis=1))
y = np.array(df['rain'])

x_train, x_test, y_train, y_test = train_test_split(x, y,
 test_size=0.2, random_state=1234)

# model 1

model1 = KNeighborsClassifier(n_neighbors=3)
model1.fit(x_train, y_train)
y_pred = model1.predict(x_test)
accuracy = model1.score(x_test, y_test)
print('accuracy:', accuracy)

cm = confusion_matrix(y_pred, y_test)
classes = [0, 1]
plt.figure()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues
)
plt.title('Random Forest')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
thresh = cm.max() / 2
for i, j in itertools.product(range(cm.shape[0]), range(c
m.shape[1])):
    plt.text(j, i, cm[i, j], horizontalalignment="center"
, color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
```

```
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

classification_report(y_test, y_pred)

# model 2

model2 = KNeighborsClassifier(n_neighbors=5)
model2.fit(x_train, y_train)
y_pred2 = model1.predict(x_test)
accuracy = model1.score(x_test, y_test)
print('accuracy:', accuracy)

cm = confusion_matrix(y_pred2, y_test)
classes = [0, 1]
plt.figure()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues
)
plt.title('Random Forest')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
thresh = cm.max() / 2
for i, j in itertools.product(range(cm.shape[0]), range(c
m.shape[1])):
    plt.text(j, i, cm[i, j], horizontalalignment="center"
, color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

classification_report(y_test, y_pred2)
```