1. Image dataset

**Model Description (algorithms)**

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

1. Normal CNN model.
2. CNN model with augmented data by rotating, shifting and zooming in the original data to generate new data.
3. CNN model trained with less amount of data.

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

一張含有 文字 的圖片

自動產生的描述

**Dataset Description**

* [Intel image Classification dataset from Kaggle](https://www.kaggle.com/datasets/puneet6060/intel-image-classification)

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

**Training Process**

**一張含有 文字 的圖片

自動產生的描述** Model A:

一張含有 桌 的圖片

自動產生的描述Model B:

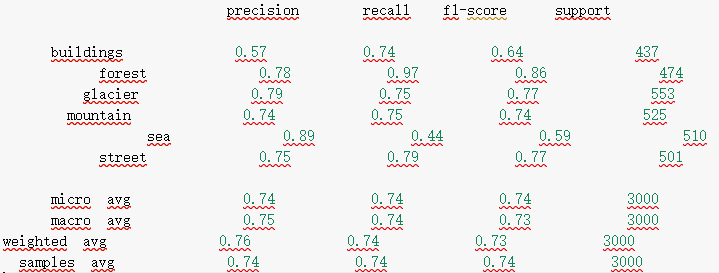
一張含有 桌 的圖片

自動產生的描述Model C:

**Performances**

Model A:**一張含有 桌 的圖片

自動產生的描述**

Model B:****

Model C:**一張含有 桌 的圖片

自動產生的描述**

**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.

1. Non-Image dataset

**Model Description (algorithms)**

* Using RandomForest and Adaboost with ExtraTreeClassifier:

1. RandomForest: n\_estimators=3 and max\_depth=100 (best result after adjusting the hyperparameters.)
2. Adaboost using Extratree Classifier.

**Dataset Description**

* [Loan prediction dataset from Univ.AI hackathon contest](https://hack.univ.ai/hackathon/1/problem-data)

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.

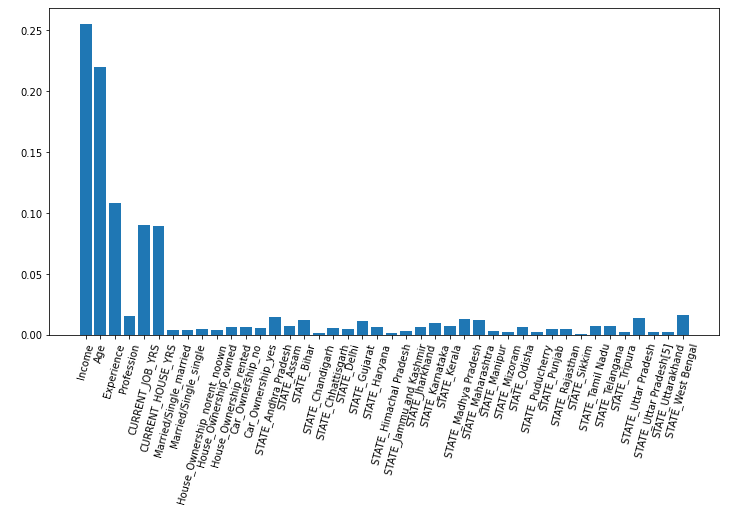
**Data Preprocessing**

* 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.

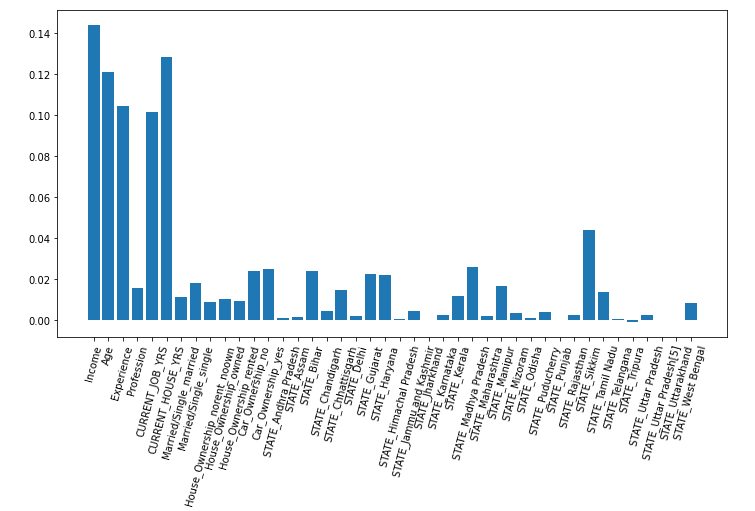
**Training Process**

* 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:



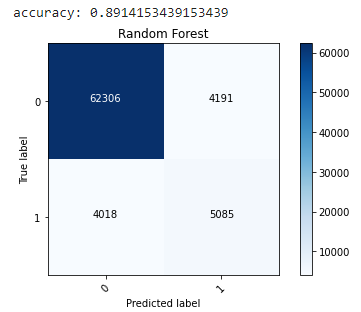
Model B:



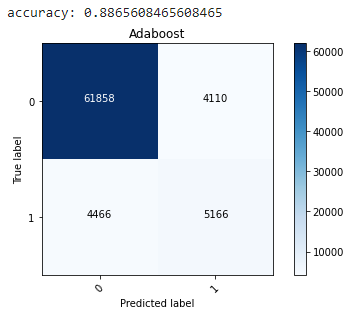
**Performances**

* Performance is measured by accuracy and confusion matrix.

Model A:



Model B:



**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.

1. Self-made dataset

**Model Description (algorithms)**

* Using KNN with 2 different hyper parameters:

1. N\_neighbors = 3
2. N\_neighbors = 5

**Dataset Description**

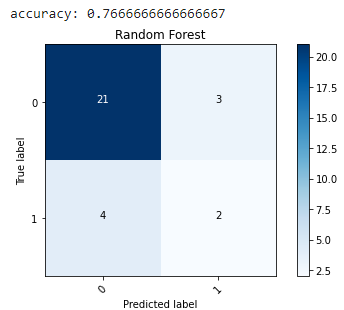
* Using part of data collected from CWB(中央氣象局) and label them by myself. I collected information about avg, highest and lowest temperature, avg, min humidity every day from 2022/10 to 2023/2. The goal is to predict whether the day will rain or not. So I also search the precipitation for each day to label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

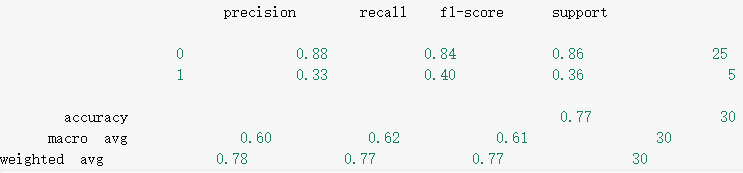
**Data Preprocessing**

* 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.

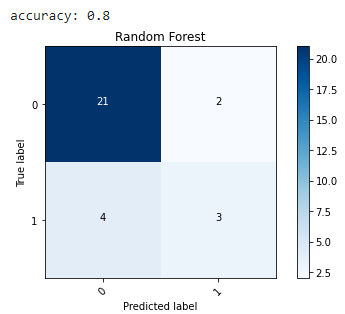
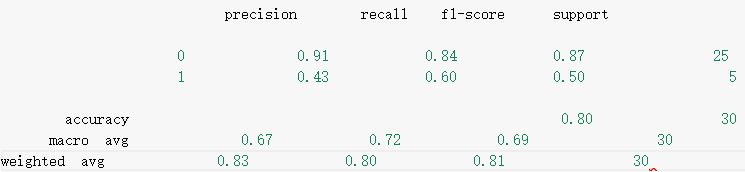
**Performances**

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

Model A:



Model B:



**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.

1. 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

1. 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 reduce 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 = '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)

])

model1.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics=['accuracy'])

model1.fit(x\_train, y\_train, batch\_size=64, epochs=20, validation\_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=['buildings', '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 = '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)

])

model2.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics=['accuracy'])

model2.fit(train\_datagen.flow(x\_train, y\_train, batch\_size=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=['buildings', '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 = '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)

])

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\_crossentropy', metrics=['accuracy'])

model3.fit(x\_train\_less, y\_train\_less, batch\_size=64, epochs=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=['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street'])

1. 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']:

  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', 'House\_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\_numpy()

y\_train = train['Risk\_Flag'].to\_numpy()

x\_test = test.to\_numpy()

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x\_train, y\_train, test\_size=0.3, random\_state=1234)

# train models

clf = RandomForestClassifier(n\_estimators=3, max\_depth=100)

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(cm.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(cm.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=10)

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(cm.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, classification\_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(cm.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(cm.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)