AI Capstone HW1

Datasets

1. Public image dataset: Intel Image Classification Dataset

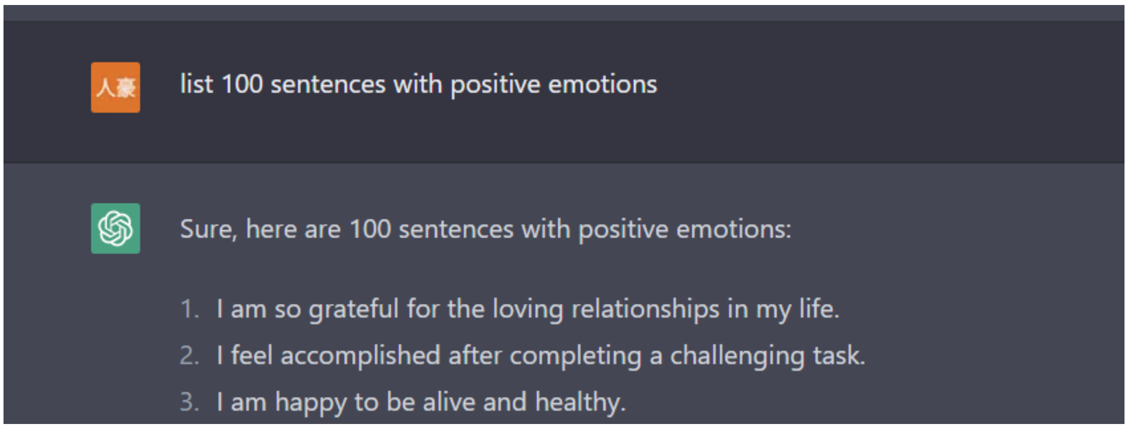
   - Introduction: Images of Natural Scenes around the world.
   - Content: It contains around 14k training images and 3k testing images of size 150x150 distributed under 6 categories.
   - Classes and their labels:
     1. buildings -> 0
     2. forest -> 1
     3. glacier -> 2
     4. imountain -> 3
     5. sea -> 4
     6. street -> 5

2. Public non-image dataset: Titanic Dataset

   - Introduction: It is a dataset for Titanic ML competition, which use machine learning to create a model that predicts which passengers survived the Titanic shipwreck.
   - Content: It contains 892 records of passengers' information and the ground truths of survive or not.
   - Features:
     1. PassengerId: Passengers' ID
     2. Pclass: Ticket class. 1st = Upper, 2nd = Middle, 3rd = Lower
     3. Name: Passengers' name
     4. Sex: Passengers' sex
     5. Age: Passengers' age (in years)
     6. SibSp: The number of siblings and spouses aboard the Titanic
     7. Parch: The number of parents and children aboard the Titanic
     8. Ticket: Ticket number
     9. Fare: Passenger fare
     10. Cabin: Cabin number
     11. Embarked: Port of Embarkation. C = Cherbourg, Q = Queenstown, S = Southampton
   - Classes and labels:
     - Survived: 1 = survived, 0 = death

3. Self-made dataset: Sentiment Analysis Dataset

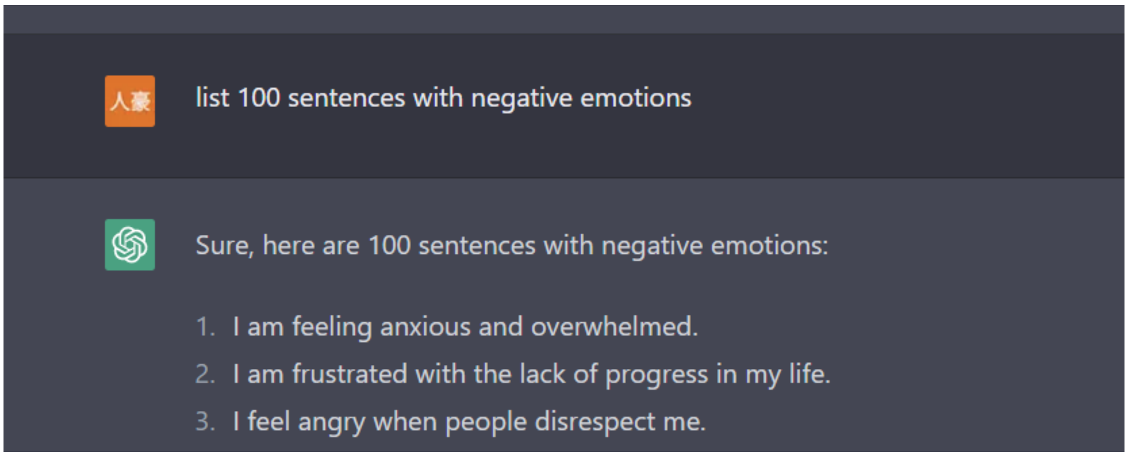
   - Introduction: It is a self-made text dataset for Sentiment Analysis.
   - Content: It contains 100 sentences with positive emotion and another 100 sentences with negative emotion.
   - Data collection process:
     - For 100 sentences with positive emotion: Tell ChatGPT to list 100 sentences with positive emotion, just like the following picture.


人豪 list 100 sentences with positive emotions

Sure, here are 100 sentences with positive emotions:

1. I am so grateful for the loving relationships in my life.
2. I feel accomplished after completing a challenging task.
3. I am happy to be alive and healthy.

- For 100 sentences with negative emotion: Tell ChatGPT to list 100 sentences with negative emotion, just like the following picture.


人豪 list 100 sentences with negative emotions

Sure, here are 100 sentences with negative emotions:

1. I am feeling anxious and overwhelmed.
2. I am frustrated with the lack of progress in my life.
3. I feel angry when people disrespect me.

- Labeling: Put 100 sentences with positive emotion in `positive_corpus.txt` and put 100 sentences with negative emotion in `negative_corpus.txt`.

Dataset Preprocessing

- For Intel Image Classification Dataset:

  1. Rescale pixel value of images from 0~255 to 0~1

- For Titanic Dataset:

  1. Get target value from 'Survived' and delete it from X
  2. Delete column 'PassengerId', 'Name', and 'Ticket' because they don't seems like a beneficial feature
  3. Delete column 'Age' and 'Cabin' because they have too much nan values
  4. Fill the nan values of 'Embarked' with the most popular value
  5. Map the categorical features of 'sex' and 'Embarked' to number
  6. Add a column 'FamilySize' from column 'SibSp' and 'Parch'
  7. Add a column 'IsAlone' from column 'FamilySize'

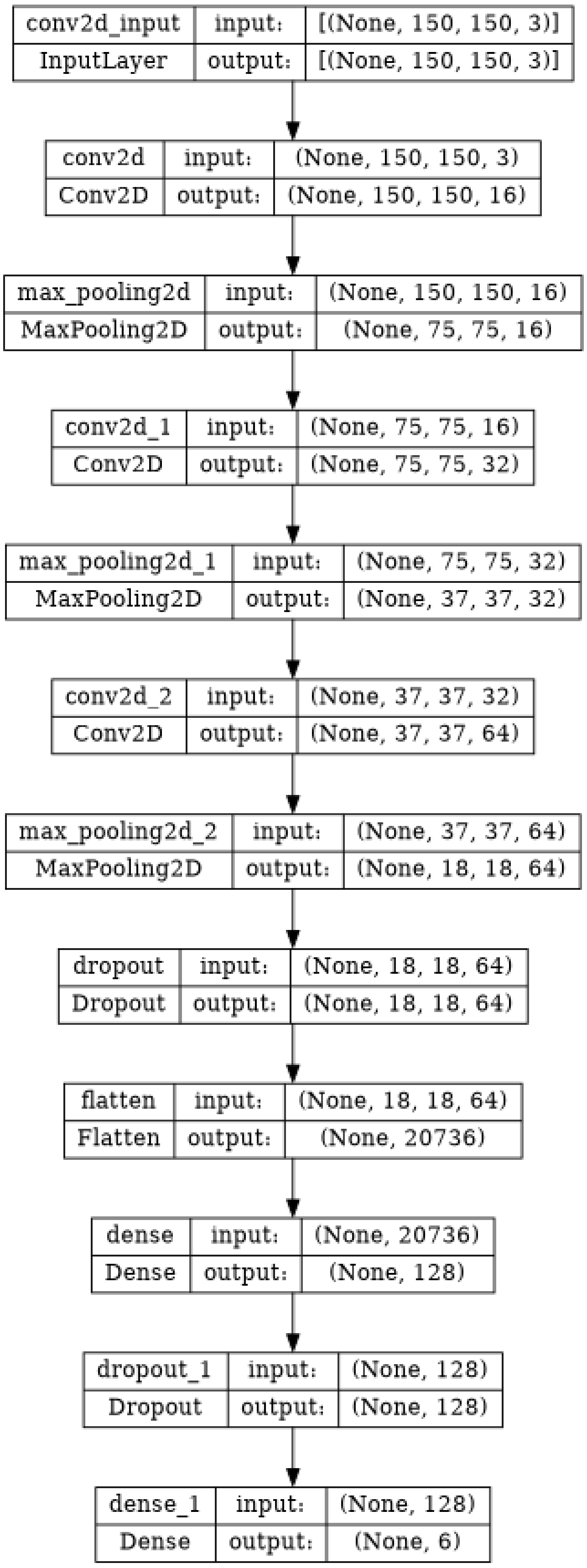
- For Sentiment Analysis Dataset:

  1. Tokenize corpus
  2. Do stemming to words
  3. Use `TfidfVectorizer` to fit training set and vectorize training and testing set
  4. Chooses to use PCA or not

Algorithm

- For Intel Image Classification:

  1. CNN: A normal convolution neural network with three convolution layers, two maxpooling layers, two dropout layers, and two dense layers. The following image is its architecture.


conv2d_input | input: | [(None, 150, 150, 3)]
InputLayer | output: | [(None, 150, 150, 3)]
conv2d | input: | (None, 150, 150, 3)
Conv2D | output: | (None, 150, 150, 16)
max_pooling2d | input: | (None, 150, 150, 16)
MaxPooling2D | output: | (None, 75, 75, 16)
conv2d_1 | input: | (None, 75, 75, 16)
Conv2D | output: | (None, 75, 75, 32)
max_pooling2d_1 | input: | (None, 75, 75, 32)
MaxPooling2D | output: | (None, 37, 37, 32)
conv2d_2 | input: | (None, 37, 37, 32)
Conv2D | output: | (None, 37, 37, 64)
max_pooling2d_2 | input: | (None, 37, 37, 64)
MaxPooling2D | output: | (None, 18, 18, 64)
dropout | input: | (None, 18, 18, 64)
Dropout | output: | (None, 18, 18, 64)
flatten | input: | (None, 18, 18, 64)
Flatten | output: | (None, 20736)
dense | input: | (None, 20736)
Dense | output: | (None, 128)
dropout_1 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 6)

2. Transfer Learning from VGG16: Get pretrained VGG16 model without the last layer from tensorflow keras, let the pretrained model untrainable, and add one flatten layer, one dropout layer, and two dense layers to the model.

- For Titanic Dataset:

    1. KNN: K-nearest neighbor (KNN) is a simple and versatile algorithm used for both classification and regression analysis. In KNN, the class or value of a new data point is predicted based on the K closest data points in the training set.
    2. SVC: Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. Support Vector Classifier (SVC) uses SVM for classification problems.
    3. Catboost: Catboost is developed by Yandex, a Russia search engine company. It is a machine learning algorithm which uses Gradient Boosting to design, and Decision Tree as its backbone. I choose it because it has two following advantages:
        - The trees inside Catboost are symmetric, which have higher efficiency and can avoid a part of overfitting
        - Can include categorical features in the dataset

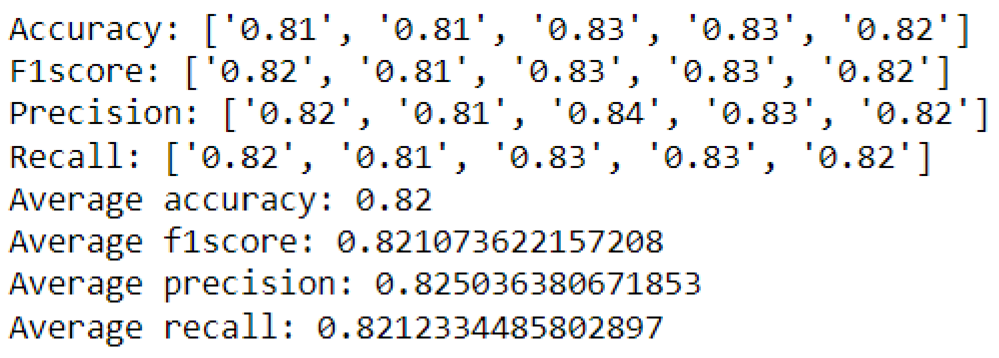
- For Sentiment Analysis Dataset:

    1. Gaussian Naive Bayes: Gaussian Naive Bayes is a probabilistic classification algorithm that is based on Bayes' theorem. It is a simple and efficient algorithm that is often used in text classification and spam filtering. It assumes that the distribution of the features is Gaussian (normal distribution).
    2. Bernoulli Naive Bayes: Bernoulli Naive Bayes is also a probabilistic classification algorithm that is based on Bayes' theorem. It is similar to Gaussian Naive Bayes, but it assumes that the features are binary and that the distribution of the features is Bernoulli.
    3. Catboost

Analysis

- For Intel Image Classification:

    1. CNN: The following metrics are the results of 5-fold cross validation on test datasets.



```
Accuracy: ['0.81', '0.81', '0.83', '0.83', '0.82']
F1score: ['0.82', '0.81', '0.83', '0.83', '0.82']
Precision: ['0.82', '0.81', '0.84', '0.83', '0.82']
Recall: ['0.82', '0.81', '0.83', '0.83', '0.82']
Average accuracy: 0.82
Average f1score: 0.821073622157208
Average precision: 0.825036380671853
Average recall: 0.8212334485802897
```

    2. Transfer Learning from VGG16: The following metrics are the results of 5-fold cross validation on test datasets.

```
Accuracy: ['0.88', '0.88', '0.88', '0.87', '0.85']
F1score: ['0.88', '0.89', '0.88', '0.88', '0.85']
Precision: ['0.88', '0.89', '0.89', '0.88', '0.88']
Recall: ['0.88', '0.89', '0.88', '0.88', '0.85']
Average accuracy: 0.8735333333333333
Average f1score: 0.8759250985478586
Average precision: 0.8823271380573475
Average recall: 0.8764593897398842
```

- From the above results, we can see Transfer Learning from VGG16 has better performance than CNN.

- For Titanic Dataset:

  - In this dataset, I use `GridSearchCV` from scikit learn to select best parameters with 5-fold cross validation.

  1. KNN: The following metrics are the results after 5-fold cross validation of `GridSearchCV` on test datasets.
     - Accuracy: 0.85
     - F1_score: 0.83
     - Precision: 0.84
     - Recall: 0.82
  2. SVC: The following metrics are the results after 5-fold cross validation of `GridSearchCV` on test datasets.
     - Accuracy: 0.84
     - F1_score: 0.81
     - Precision: 0.86
     - Recall: 0.79
  3. Catboost: The following metrics are the results after 5-fold cross validation of `GridSearchCV` on test datasets.
     - Without one-hot encoding and standardize:
       - Accuracy: 0.83
       - F1_score: 0.81
       - Precision: 0.83
       - Recall: 0.80
     - With one-hot encoding and standardize:
       - Accuracy: 0.85
       - F1_score: 0.83
       - Precision: 0.85
       - Recall: 0.82

  - From the above results, we can see three model have similar performances.

- For Sentiment Analysis Dataset:

  - In this dataset, I also use `GridSearchCV` from scikit learn to select best parameters with 5-fold cross validation.

1. Gaussian Naive Bayes: The following metrics are the results after 5-fold cross validation of `GridSearchCV` on test datasets.

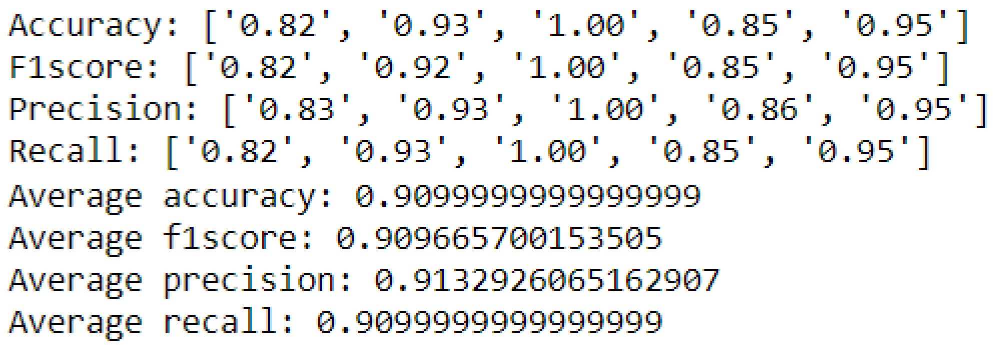
   - With PCA:
     ```
     Accuracy: ['0.95', '0.95', '0.90', '0.93', '0.85']
     F1score: ['0.95', '0.95', '0.90', '0.92', '0.85']
     Precision: ['0.95', '0.95', '0.92', '0.93', '0.88']
     Recall: ['0.95', '0.95', '0.90', '0.93', '0.85']
     Average accuracy: 0.9149999999999998
     Average f1score: 0.9139724651061432
     Average precision: 0.9290311138137227
     Average recall: 0.9149999999999998
     ```
   - Without PCA:
     ```
     Accuracy: ['0.95', '1.00', '1.00', '1.00', '1.00']
     F1score: ['0.95', '1.00', '1.00', '1.00', '1.00']
     Precision: ['0.95', '1.00', '1.00', '1.00', '1.00']
     Recall: ['0.95', '1.00', '1.00', '1.00', '1.00']
     Average accuracy: 0.99
     Average f1score: 0.9899749373433584
     Average precision: 0.990909090909091
     Average recall: 0.99
     ```

2. Bernoulli Naive Bayes: The following metrics are the results after 5-fold cross validation of `GridSearchCV` on test datasets.

   - With PCA:
     ```
     Accuracy: ['0.88', '0.57', '1.00', '0.65', '0.62']
     F1score: ['0.87', '0.54', '1.00', '0.63', '0.59']
     Precision: ['0.88', '0.61', '1.00', '0.70', '0.68']
     Recall: ['0.88', '0.57', '1.00', '0.65', '0.62']
     Average accuracy: 0.745
     Average f1score: 0.7272277729361784
     Average precision: 0.7725356401757081
     Average recall: 0.745
     ```
   - Without PCA:
     ```
     Accuracy: ['0.97', '1.00', '1.00', '1.00', '1.00']
     F1score: ['0.97', '1.00', '1.00', '1.00', '1.00']
     Precision: ['0.98', '1.00', '1.00', '1.00', '1.00']
     Recall: ['0.97', '1.00', '1.00', '1.00', '1.00']
     Average accuracy: 0.9949999999999999
     Average f1score: 0.9949968730456537
     Average precision: 0.9952380952380953
     Average recall: 0.9949999999999999
     ```

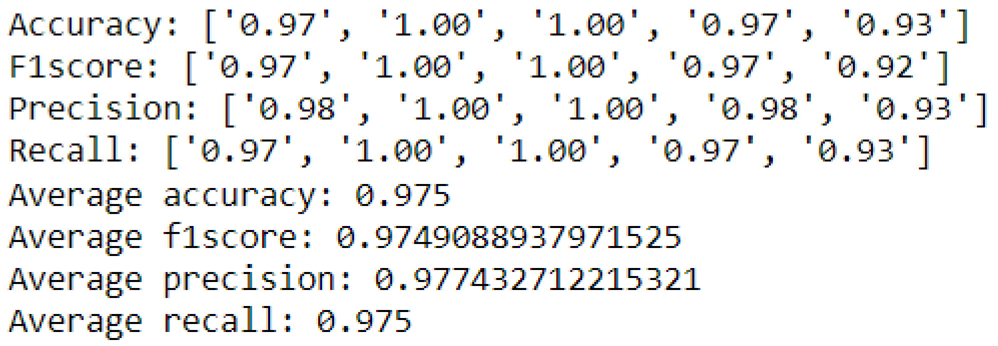
3. Catboost: The following metrics are the results after 5-fold cross validation of `GridSearchCV` on test datasets.

- With PCA:



```
Accuracy: ['0.82', '0.93', '1.00', '0.85', '0.95']
F1score: ['0.82', '0.92', '1.00', '0.85', '0.95']
Precision: ['0.83', '0.93', '1.00', '0.86', '0.95']
Recall: ['0.82', '0.93', '1.00', '0.85', '0.95']
Average accuracy: 0.9099999999999999
Average f1score: 0.909665700153505
Average precision: 0.9132926065162907
Average recall: 0.9099999999999999
```

- Without PCA:



```
Accuracy: ['0.97', '1.00', '1.00', '0.97', '0.93']
F1score: ['0.97', '1.00', '1.00', '0.97', '0.92']
Precision: ['0.98', '1.00', '1.00', '0.98', '0.93']
Recall: ['0.97', '1.00', '1.00', '0.97', '0.93']
Average accuracy: 0.975
Average f1score: 0.9749088937971525
Average precision: 0.977432712215321
Average recall: 0.975
```

- From the above results, we can see all three models have good performance on dataset without PCA, but perform worse on dataset with PCA.

Discussion

- From all the experiment and results above, I discovered something interesting:
  - Cross Validation is really important, especially on dataset with small size. For example, because Sentiment Analysis Dataset has only 200 records in total, so the performances differ a lot.
  - PCA not always increase the performance. In Sentiment Analysis Dataset, because its feature vector size is 500, but it has only 160 records in training set, so the best `n_component` for PCA is 120. However, to cut feature size from 500 to 120 is too much, so the models performed worse on dataset with PCA.
- For future work if there were more time available, I would like to dive more deep into the datasets. Having more analysis, feature engineering and preprocessing will help the model performs better. In addition to that, I would like to try other algorithms and try more hyper-parameters.

Appendix: Code

- For Intel Image Classification:

```
import numpy as np
import pandas as pd
import tensorflow as tf
import albumentations as A
import matplotlib.pyplot as plt
import tensorflow_addons as tfa
```

```
from tensorflow import keras
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.preprocessing.image import array_to_img
from tensorflow.keras.utils import image_dataset_from_directory, plot_model
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, confusion_matrix, classification_report
from tensorflow.keras.layers import Rescaling, RandomFlip, RandomRotation,
RandomZoom, Dense, Flatten, Dropout, Conv2D, MaxPooling2D

# !kaggle datasets download -d puneet6060/intel-image-classification
# from zipfile import ZipFile
# file_name="./dataset/intel-image-classification.zip"

# with ZipFile(file_name,'r') as zip:
#     zip.extractall()

# Set dataset path
train_path = './dataset/seg_train/seg_train/'
test_path = './dataset/seg_test/seg_test/'

'''
CNN Model
'''

def get_CNN_model():
    # CNN model architecture
    model = Sequential([
        Conv2D(16, 3, padding='same', activation='relu', input_shape=(150,
150, 3)),
        MaxPooling2D(),
        Conv2D(32, 3, padding='same', activation='relu'),
        MaxPooling2D(),
        Conv2D(64, 3, padding='same', activation='relu'),
        MaxPooling2D(),
        Dropout(0.3),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.4),
        Dense(6, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Load dataset to type tensorflow PrefetchDataset
def load_train_validation_dataset(path, seed):
    train_ds = image_dataset_from_directory(
        directory=path,
        seed=seed,
        validation_split=0.2,
        subset="training",
        image_size=(150, 150),
        shuffle=True,
    )
    val_ds = image_dataset_from_directory(
        directory=path,
        seed=seed,
        validation_split=0.2,
        subset="validation",
        image_size=(150, 150),
        shuffle=True,
    )
    return train_ds, val_ds

# Load dataset to type tensorflow PrefetchDataset
def load_test_dataset(path, seed):
    test_ds = image_dataset_from_directory(
        directory=path,
        image_size=(150, 150),
        shuffle=False
    )
    return test_ds

def preprocessing(train_ds, val_ds, test_ds):
    # Rescale pixel value from 0~255 to 0~1
    scaling = tf.keras.layers.Rescaling(1./255)
    train_ds = train_ds.map(lambda x, y: (scaling(x), y))
    val_ds = val_ds.map(lambda x, y: (scaling(x), y))
    test_ds = test_ds.map(lambda x, y: (scaling(x), y))

    # Make training process faster
    train_ds =
train_ds.cache().shuffle(buffer_size=64).prefetch(buffer_size=tf.data.AUTOTU
NE)
    val_ds =
val_ds.cache().shuffle(buffer_size=64).prefetch(buffer_size=tf.data.AUTOTUNE
)

    return train_ds, val_ds, test_ds

# get test dataset's label from PrefetchDataset and return numpy array
def get_test_label(test_ds):
    test_data = [(image.numpy(), label.numpy()) for image, label in test_ds]
    test_label = test_data[0][1]
    for i in range(1, len(test_data)):
        test_label = np.concatenate([test_label, test_data[i][1]], axis=0)

    return list(test_label)

# turn probability prediction to label
def get_prediction_label(prediction):
    predict_label = np.argmax(prediction, axis=1)

    return list(predict_label)
```

```
def get_metric_results(test_label, predict_label):
    metric_results = {}
    metric_results['accuracy'] = accuracy_score(test_label, predict_label)
    metric_results['f1_score'] = f1_score(test_label, predict_label,
average="macro")
    metric_results['precision'] = precision_score(test_label, predict_label,
average="macro")
    metric_results['recall'] = recall_score(test_label, predict_label,
average="macro")
    return metric_results

accuracy = []
f1score = []
precision = []
recall = []

for i in range(5):
    seed = i ** 2
    train_ds, val_ds = load_train_validation_dataset(train_path, seed)
    test_ds = load_test_dataset(test_path, seed)
    train_ds, val_ds, test_ds = preprocessing(train_ds, val_ds, test_ds)

    model = get_CNN_model()
    model.fit(train_ds,
              validation_data=val_ds,
              epochs=10)

    prediction = model.predict(test_ds)

    test_label = get_test_label(test_ds)
    predict_label = get_prediction_label(prediction)

    metric_result = get_metric_results(test_label, predict_label)

    accuracy.append(metric_result['accuracy'])
    f1score.append(metric_result['f1_score'])
    precision.append(metric_result['precision'])
    recall.append(metric_result['recall'])

print('Accuracy:', ['%.2f' % val for val in accuracy])
print('F1score:', ['%.2f' % val for val in f1score])
print('Precision:', ['%.2f' % val for val in precision])
print('Recall:', ['%.2f' % val for val in recall])
print('Average accuracy:', sum(accuracy) / len(accuracy))
print('Average f1score:', sum(f1score) / len(f1score))
print('Average precision:', sum(precision) / len(precision))
print('Average recall:', sum(recall) / len(recall))

'''
Transfer Learning VGG
'''

def get_VGG16_pretrained_model():
    pretrained_model = VGG16(input_shape = (150, 150, 3),
                             include_top = False,
                             weights = 'imagenet')

    for layer in pretrained_model.layers:
         layer.trainable = False

    return pretrained_model

def get_transfered_model(last_layer_output, pretrained_model_input):
    x = Flatten()(last_layer_output)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(6, activation='softmax')(x)
    model = Model(pretrained_model_input, x)

    model.compile(optimizer =
tf.keras.optimizers.RMSprop(learning_rate=0.0001),
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])

    return model

VGG_accuracy = []
VGG_f1score = []
VGG_precision = []
VGG_recall = []

pretrained_model = get_VGG16_pretrained_model()
last_layer = pretrained_model.get_layer('block5_pool')

for i in range(5):
    seed = i ** 2
    train_ds, val_ds = load_train_validation_dataset(train_path, seed)
    test_ds = load_test_dataset(test_path, seed)
    train_ds, val_ds, test_ds = preprocessing(train_ds, val_ds, test_ds)

    model = get_transfered_model(last_layer.output, pretrained_model.input)

    model.fit(train_ds,
              validation_data=val_ds,
              epochs=5)

    prediction = model.predict(test_ds)

    test_label = get_test_label(test_ds)
    predict_label = get_prediction_label(prediction)

    metric_result = get_metric_results(test_label, predict_label)

    VGG_accuracy.append(metric_result['accuracy'])
    VGG_f1score.append(metric_result['f1_score'])
    VGG_precision.append(metric_result['precision'])
    VGG_recall.append(metric_result['recall'])
```

```
print('Accuracy:', ['%.2f' % val for val in VGG_accuracy])
print('F1score:', ['%.2f' % val for val in VGG_f1score])
print('Precision:', ['%.2f' % val for val in VGG_precision])
print('Recall:', ['%.2f' % val for val in VGG_recall])
print('Average accuracy:', sum(VGG_accuracy) / len(VGG_accuracy))
print('Average f1score:', sum(VGG_f1score) / len(VGG_f1score))
print('Average precision:', sum(VGG_precision) / len(VGG_precision))
print('Average recall:', sum(VGG_recall) / len(VGG_recall))
```

- For Titanic Dataset:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from catboost import Pool, CatBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, confusion_matrix, classification_report

'''
Preprocess Dataset
'''

# Read csv file
df = pd.read_csv('./dataset/train.csv')

# Get target value from 'Survived'
Y = df["Survived"].values

# Delete column 'PassengerId', 'Name', and 'Ticket' because they don't seems
like a beneficial feature
df = df.drop(["PassengerId", "Survived", "Name", "Ticket"], axis=1)
df.head()

# Get nan counts for each feature
print(df.isna().sum())

# Delete column 'Age' and 'Cabin' because they have too much nan values
df = df.drop(["Age", "Cabin"], axis=1)

# Show the distribution of 'Embarked'
embarked_info = df["Embarked"].value_counts()
plt.bar(embarked_info.index, embarked_info.values)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Embarked distribution", fontsize=16);
```

```
# Fill the nan values of 'Embarked' with the most popular value
df["Embarked"].fillna("S", inplace=True)

# Map the categorical features of 'sex' and 'Embarked' to number
df["Sex"] = df["Sex"].map({"male": 1, "female": 0}).astype(int)
df["Embarked"] = df["Embarked"].map({"S": 1, "C": 2, "Q": 3}).astype(int)

# Add a column 'FamilySize' from column 'SibSp' and 'Parch'
df["FamilySize"] = df["SibSp"] + df["Parch"] + 1

# Add a column 'IsAlone' from column 'FamilySize'
df["IsAlone"] = df["FamilySize"].apply(lambda x: 1 if x == 1 else 0)

# Split dataset into train and test set
train_size = int(Y.shape[0] * 0.8)
X_train = df[:train_size]
X_test = df[train_size:]
Y_train = Y[:train_size]
Y_test = Y[train_size:]

print(f"Train X shape: {X_train.shape}")
print(f"Train y shape: {Y_train.shape}")
print(f"Test X shape: {X_test.shape}")
print(f"Train y shape: {Y_test.shape}")

# Get metric scores
def get_metric_results(test_label, predict_label):
    metric_results = {}
    metric_results['accuracy'] = accuracy_score(test_label, predict_label)
    metric_results['f1_score'] = f1_score(test_label, predict_label,
average="macro")
    metric_results['precision'] = precision_score(test_label, predict_label,
average="macro")
    metric_results['recall'] = recall_score(test_label, predict_label,
average="macro")
    return metric_results

'''
Catboost with original preprocessed dataset
'''

# categorical features = ['Sex', 'Embarked', 'IsAlone']
cat_features = [1, 5, 7]

# GridSearchCV can do exhaustive search over specified parameter values for
an estimator.
# Parameters for GridSearchCV
parameters = {
    'iterations': [50, 100, 150],
    'learning_rate': [0.05, 0.1, 0.3],
    'depth': [9, 11, 13],
}
```

```
# Define classifier model
catboost_model = CatBoostClassifier(
    verbose=50,
    cat_features=cat_features,
)

# Do GridSearchCV on model
catboost_model = GridSearchCV(
    catboost_model,
    parameters,
    cv=5,
    scoring='accuracy',
)

# Train model
catboost_model.fit(X_train, Y_train)

print(catboost_model.best_params_)
print(catboost_model.best_score_)

prediction = catboost_model.predict(X_test)
print(get_metric_results(Y_test, prediction))

'''
One-hot encode and Standardize Dataset
'''

# Do one-hot encoding on 'Pclass' and 'Embarked'
one_hot_cols = ["Pclass", "Embarked"]
for col in one_hot_cols:
    df = pd.concat(
        [df, pd.get_dummies(df[col], prefix=col)],
        axis=1,
        join="inner",
    )
df = df.drop(one_hot_cols, axis=1)

# Standardize dataset
scaler = StandardScaler()
df.loc[:] = scaler.fit_transform(df)

# New train and test set after one-hot encoding and standardization
X_train_norm = df[:Y_train.shape[0]]
X_test_norm = df[Y_train.shape[0]:]

print(f"Train norm X shape: {X_train_norm.shape}")
print(f"Test norm X shape: {X_test_norm.shape}")

'''
SVC with new dataset
'''

# GridSearchCV can do exhaustive search over specified parameter values for
an estimator.
```

```
# Parameters for GridSearchCV
parameters = {
    "C": [0.001, 0.01, 0.1, 1.],
    "kernel": ["linear", "poly", "rbf", "sigmoid"],
    "gamma": ["scale", "auto"],
}

# Define classifier model
svc_model = SVC(
    random_state=42,
    class_weight="balanced",
    probability=True,
)

# Do GridSearchCV on model
svc_model = GridSearchCV(
    svc_model,
    parameters,
    cv=5,
    scoring='accuracy',
)

# Train model
svc_model.fit(X_train_norm, Y_train)

print(svc_model.best_params_)
print(svc_model.best_score_)

prediction = svc_model.predict(X_test_norm)
print(get_metric_results(Y_test, prediction))

'''
KNN with new dataset
'''

# GridSearchCV can do exhaustive search over specified parameter values for
an estimator.
# Parameters for GridSearchCV
parameters = {
    "weights": ["uniform", "distance"],
}

# Define classifier model
knn_model = KNeighborsClassifier()

# Do GridSearchCV on model
knn_model = GridSearchCV(
    knn_model,
    parameters,
    cv=5,
    scoring='accuracy',
)

# Train model
```

```
knn_model.fit(X_train_norm, Y_train)

print(knn_model.best_params_)
print(knn_model.best_score_)

prediction = knn_model.predict(X_test_norm)
print(get_metric_results(Y_test, prediction))

'''
Catboost with new dataset
'''

# GridSearchCV can do exhaustive search over specified parameter values for
an estimator.
# Parameters for GridSearchCV
parameters = {
    'iterations': [50, 100, 150],
    'learning_rate': [0.05, 0.1, 0.3],
    'depth': [9, 11, 13],
}

# Define classifier model
catboost_model = CatBoostClassifier(
    verbose=50,
    cat_features=[],
)

# Do GridSearchCV on model
catboost_model = GridSearchCV(
    catboost_model,
    parameters,
    cv=5,
    scoring='accuracy',
)

# Train model
catboost_model.fit(X_train_norm, Y_train)

print(catboost_model.best_params_)
print(catboost_model.best_score_)

prediction = catboost_model.predict(X_test_norm)
print(get_metric_results(Y_test, prediction))
```

- For Sentiment Analysis Dataset:

```

import re
import warnings
import numpy as np
import pandas as pd

from tqdm import tqdm
from bs4 import BeautifulSoup

# import nltk
# nltk.download('omw-1.4')
# nltk.download('stopwords')
# nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer

from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer

from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB, GaussianNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, confusion_matrix, classification_report

from catboost import Pool, CatBoostClassifier
%matplotlib inline

# Read positive corpus
pos_df = pd.read_fwf("positive_corpus.txt", names=['Phrase'], header=None)
pos_df['Sentiment'] = 1

# Read negative corpus
neg_df = pd.read_fwf("negative_corpus.txt", names=['Phrase'], header=None)
neg_df['Sentiment'] = 0

# TfidfVectorize dataset from pandas DataFrame, split dataset for training
and validation, and convert to numpy array
def TfidfVectorizeDataframe(X_df, Y_df, seed, use_pca=True):
    # Setup stemmer for English language
    stemmer = SnowballStemmer(language='english')

    # Function to create tokenizer
    def tokenize(text):
        return [stemmer.stem(token) for token in word_tokenize(text)]

    # Create stopword for English language
    eng_stopword = stopwords.words('english')

    # Create vectorizer
    vectorizer = TfidfVectorizer(tokenizer=tokenize,
                                 stop_words=eng_stopword,
                                 ngram_range=(1,2),
                                 max_features=500)

    # Fit training dataset in vectorizer and transform training and
```

```
validation dataset
    vectorizer.fit(X_df.Phrase)
    X_dataset = vectorizer.transform(X_df.Phrase)
    Y_dataset = Y_df.Sentiment.values

    # Split dataset for training and validation
    X_train, X_val, Y_train, Y_val = train_test_split(X_dataset, Y_dataset,
test_size=0.2, stratify=Y_dataset, random_state=seed)

    # Turn sparse matrix to numpy array
    X_train = X_train.toarray()
    X_val = X_val.toarray()

    # Use PCA to reduce feature number to increase training speed and avoid
overfitting
    # Fit training dataset in pca and transform training and validation
dataset
    if use_pca:
        pca = PCA(n_components=120)
        _ = pca.fit(X_train)

        X_train = pca.transform(X_train)
        X_val = pca.transform(X_val)

    return X_train, X_val, Y_train, Y_val

# Merge positive and negative datasets
df = pd.concat([pos_df, neg_df], axis=0)
df = df.sample(frac=1).reset_index(drop=True)

# Get X
X_df = df['Phrase'].to_frame()

# Get Y
Y_df = df['Sentiment'].to_frame()

# Get metric scores
def get_metric_results(test_label, predict_label):
    metric_results = {}
    metric_results['accuracy'] = accuracy_score(test_label, predict_label)
    metric_results['f1_score'] = f1_score(test_label, predict_label,
average="macro")
    metric_results['precision'] = precision_score(test_label, predict_label,
average="macro")
    metric_results['recall'] = recall_score(test_label, predict_label,
average="macro")
    return metric_results

'''
Bernoulli Naive Bayes
'''

accuracy = []
f1score = []
```

```
precision = []
recall = []

for i in range(5):

    # Get train and test dataset after preprocessing
    X_train, X_test, Y_train, Y_test = TfidfVectorizeDataframe(X_df, Y_df,
seed=i**2, use_pca=False)

    # GridSearchCV can do exhaustive search over specified parameter values
for an estimator.
    # Parameters for GridSearchCV
    parameters = {
        'alpha': [0.6, 0.8, 1.0, 1.2, 1.4],
        'binarize': [0.0, 0.1, 0.2],
    }

    # Define Bernoulli Naive Bayes model
    nb_clf = BernoulliNB()

    # Do GridSearchCV on model
    nb_clf = GridSearchCV(
        nb_clf,
        parameters,
        cv=5,
        scoring='accuracy',
        refit=True,
    )

    # Train model
    nb_clf.fit(X_train, Y_train)

    Y_predicted = nb_clf.predict(X_test)
    metric_result = get_metric_results(Y_test, Y_predicted)

    accuracy.append(metric_result['accuracy'])
    f1score.append(metric_result['f1_score'])
    precision.append(metric_result['precision'])
    recall.append(metric_result['recall'])

print('Accuracy:', ['%.2f' % val for val in accuracy])
print('F1score:', ['%.2f' % val for val in f1score])
print('Precision:', ['%.2f' % val for val in precision])
print('Recall:', ['%.2f' % val for val in recall])
print('Average accuracy:', sum(accuracy) / len(accuracy))
print('Average f1score:', sum(f1score) / len(f1score))
print('Average precision:', sum(precision) / len(precision))
print('Average recall:', sum(recall) / len(recall))

'''
Gaussian Naive Bayes
'''

accuracy = []
```

```
f1score = []
precision = []
recall = []

for i in range(5):

    # Get train and test dataset after preprocessing
    X_train, X_test, Y_train, Y_test = TfidfVectorizeDataframe(X_df, Y_df,
seed=i**2, use_pca=False)

    # GridSearchCV can do exhaustive search over specified parameter values
for an estimator.
    # Parameters for GridSearchCV
    parameters = {
        'var_smoothing': np.logspace(0, -9, num=100),
    }

    # Define Gaussian Naive Bayes model
    nb_clf = GaussianNB()

    # Do GridSearchCV on model
    nb_clf = GridSearchCV(
        nb_clf,
        parameters,
        cv=5,
        scoring='accuracy',
        refit=True,
    )

    # Train model
    nb_clf.fit(X_train, Y_train)

    Y_predicted = nb_clf.predict(X_test)
    metric_result = get_metric_results(Y_test, Y_predicted)

    accuracy.append(metric_result['accuracy'])
    f1score.append(metric_result['f1_score'])
    precision.append(metric_result['precision'])
    recall.append(metric_result['recall'])

print('Accuracy:', ['%.2f' % val for val in accuracy])
print('F1score:', ['%.2f' % val for val in f1score])
print('Precision:', ['%.2f' % val for val in precision])
print('Recall:', ['%.2f' % val for val in recall])
print('Average accuracy:', sum(accuracy) / len(accuracy))
print('Average f1score:', sum(f1score) / len(f1score))
print('Average precision:', sum(precision) / len(precision))
print('Average recall:', sum(recall) / len(recall))

'''
Catboost
'''

accuracy = []
```

```
f1score = []
precision = []
recall = []

for i in range(5):

    # Get train and test dataset after preprocessing
    X_train, X_test, Y_train, Y_test = TfidfVectorizeDataframe(X_df, Y_df,
seed=i**2, use_pca=False)

    # Fit directly without GridSearchCV

    # Build train and validation dataset for Catboost
    train_dataset = Pool(data=X_train,
                         label=Y_train,
                         cat_features=[])

    eval_dataset = Pool(data=X_test,
                        label=Y_test,
                        cat_features=[])

    # Get CatBoostClassifier model
    model = CatBoostClassifier(iterations=4000,
                               learning_rate=0.3,
                               l2_leaf_reg=5,
                               depth=10,
                               use_best_model=True,
                               early_stopping_rounds=300,
                               task_type="GPU",
                               devices='0:1')
    # Fit model
    model.fit(train_dataset, eval_set=eval_dataset,
early_stopping_rounds=25, verbose=25, plot=True)

    Y_predicted = model.predict(X_test)
    metric_result = get_metric_results(Y_test, Y_predicted)

    accuracy.append(metric_result['accuracy'])
    f1score.append(metric_result['f1_score'])
    precision.append(metric_result['precision'])
    recall.append(metric_result['recall'])

print('Accuracy:', ['%.2f' % val for val in accuracy])
print('F1score:', ['%.2f' % val for val in f1score])
print('Precision:', ['%.2f' % val for val in precision])
print('Recall:', ['%.2f' % val for val in recall])
print('Average accuracy:', sum(accuracy) / len(accuracy))
print('Average f1score:', sum(f1score) / len(f1score))
print('Average precision:', sum(precision) / len(precision))
print('Average recall:', sum(recall) / len(recall))
```