MeowTech presents you…

Song & book recommendation using audio sentiment analysis

- Debajyoti Sarkar
- Priyanshu Sarkar
- Chirantan Das
- Rohan Sharma

Introduction

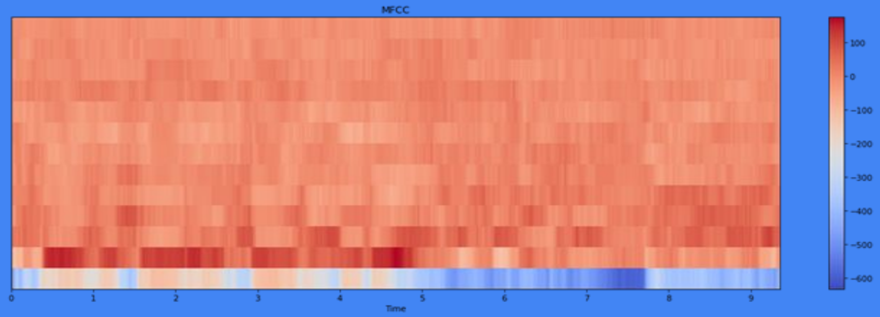
This project revolves around creating a Software which will recommend you songs and books based on your emotions derived from your recorded audio sample, using KNN ,XGB and MLP models. The project is still a work in progress. The details of the project are described in further slides.

Neural network

Neural networks are a type of machine learning algorithm modeled after the human brain that can be used to analyze and process audio data. They can identify patterns and features that are difficult for humans to detect, such as different phonemes in speech or musical notes in a piece of music. One of the advantages of using neural networks in audio analysis is their ability to learn and adapt over time.

They can be trained on large amounts of data and improve their accuracy and performance with more examples. Neural networks have become an increasingly popular tool in audio analysis, allowing researchers and developers to create more advanced and sophisticated audio processing systems. As technology continues to evolve, it is likely that we will see even more innovative uses of neural networks in audio analysis in the future.

Audio analysis



Audio analysis is the process of analyzing and understanding the characteristics of sound signals. It involves using various techniques and methods to extract meaningful information from audio data, such as speech, music, and other types of sounds

The goal of audio analysis is to gain insights into the content and structure of audio signals, and to use that information for various purposes such as speech recognition, music recommendation, and even emotion detection. Audio analysis can help us better understand how sound works and how it affects us, and can be applied in many different industries and fields.

TECHNOLOGIES USED

- 🐱Intel Python3
- 🐱Librosa (for Feature Extraction)
- 🐱Intel optimized scikit learn

Progress

✅creating intel python3 env

✅data cleaning(from RAVDESS & TESS)

✅feature extraction from clean dataset

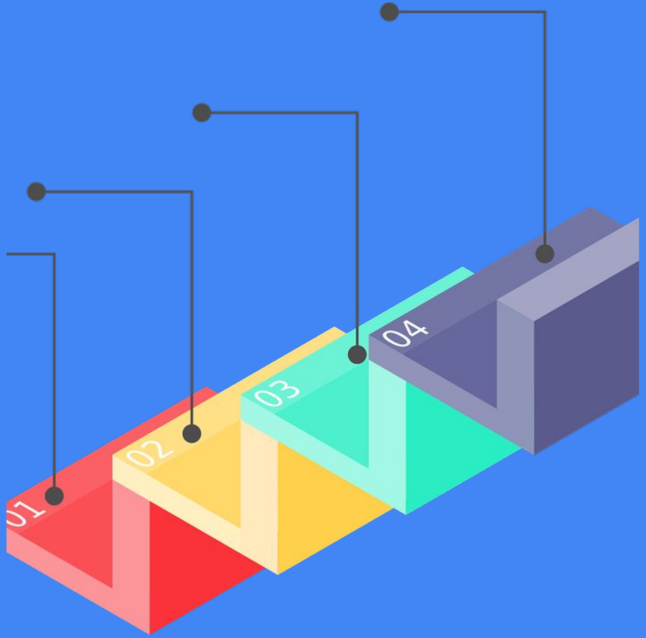
✅dimensional reduction(via PCA)

✅Implementing K-means Nearest Neighbour algorithm

✅Implementing XGBoost

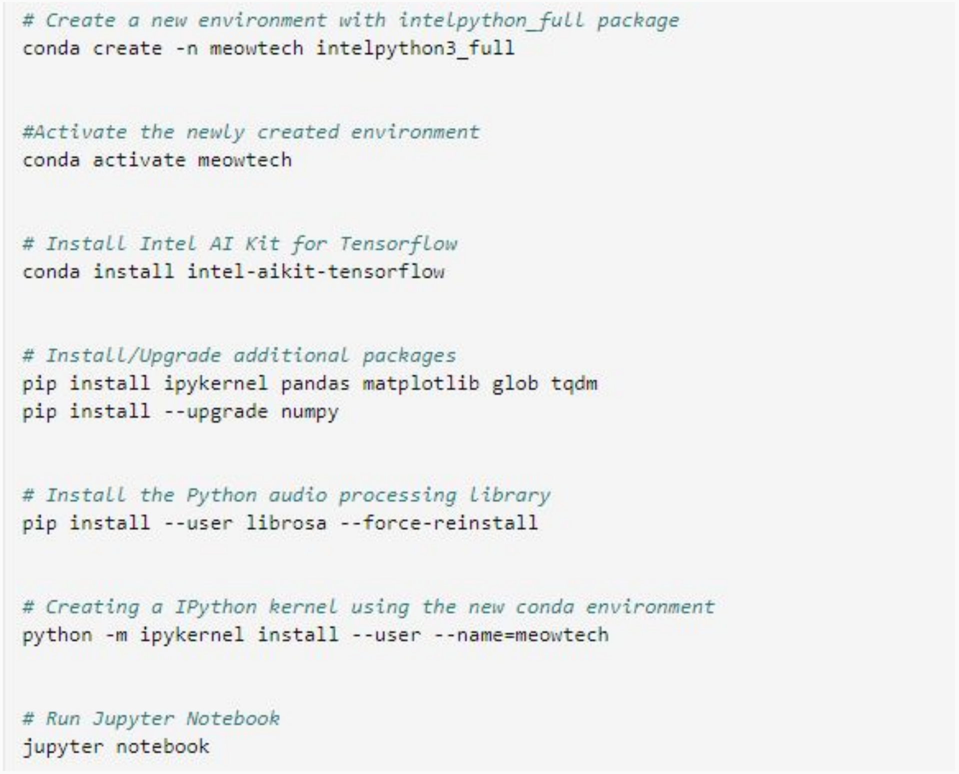
✅Implementing Multi Layer Perceptron

✅Ensembling the 3 models together



Creating intel python3 env

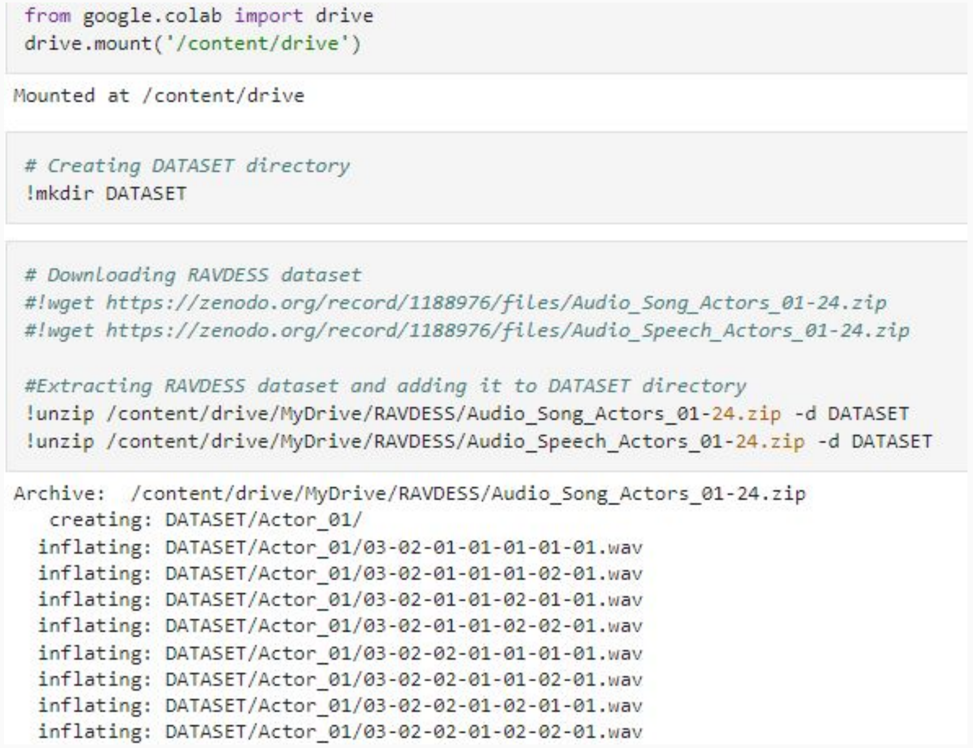
Intel Python3 is a distribution of Python optimized for performance on Intel processors. An Intel Python 3 environment is a virtual environment that uses the Intel distribution to isolate and optimize Python code. This environment can significantly improve the speed and efficiency of Python code on Intel hardware, making it useful for computationally intensive tasks like scientific computing and data analysis.



```
# Create a new environment with intelpython_full package
conda create -n meowtech intelpython3_full


#Activate the newly created environment
conda activate meowtech


# Install Intel AI Kit for Tensorflow
conda install intel-aikit-tensorflow


# Install/Upgrade additional packages
pip install ipykernel pandas matplotlib glob tqdm
pip install --upgrade numpy


# Install the Python audio processing library
pip install --user librosa --force-reinstall


# Creating a IPython kernel using the new conda environment
python -m ipykernel install --user --name=meowtech


# Run Jupyter Notebook
jupyter notebook
```

Data cleaning

Data cleaning is the process of identifying and correcting or removing errors, inconsistencies, and inaccuracies in datasets. It is a critical step in the data preparation process, as it ensures that the data is accurate and reliable before analysis. Data cleaning involves various tasks such as handling missing data, dealing with duplicates, correcting inconsistent data, and removing outliers. The goal is to ensure that the data is in a format that is suitable for analysis, which involves ensuring that it is complete, consistent, and correct. Effective data cleaning can improve the accuracy and reliability of insights derived from the data, leading to better decision-making.



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

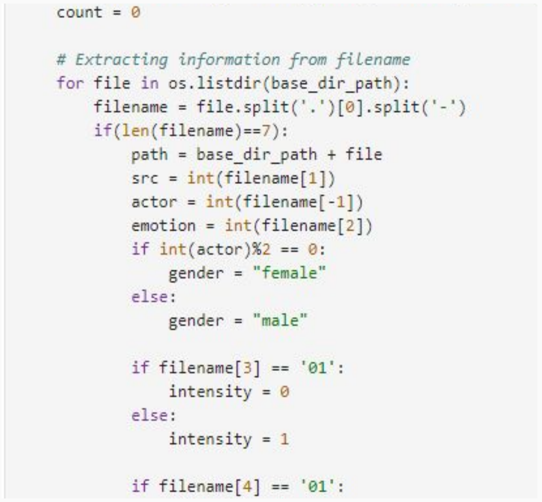
```
# Creating DATASET directory
!mkdir DATASET
```

```
# Downloading RAVDESS dataset
#!wget https://zenodo.org/record/1188976/files/Audio_Song_Actors_01-24.zip
#!wget https://zenodo.org/record/1188976/files/Audio_Speech_Actors_01-24.zip

#Extracting RAVDESS dataset and adding it to DATASET directory
!unzip /content/drive/MyDrive/RAVDESS/Audio_Song_Actors_01-24.zip -d DATASET
!unzip /content/drive/MyDrive/RAVDESS/Audio_Speech_Actors_01-24.zip -d DATASET
```

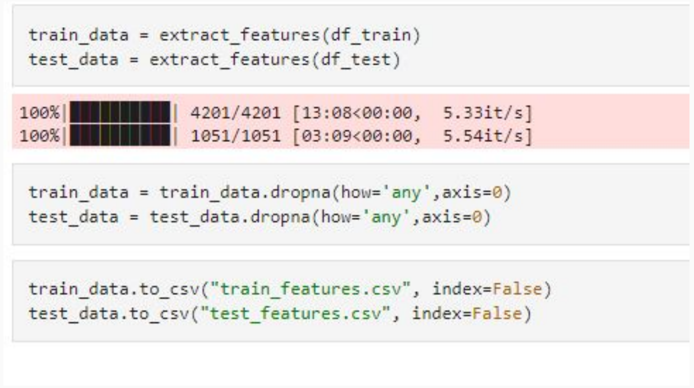
```
Archive:  /content/drive/MyDrive/RAVDESS/Audio_Song_Actors_01-24.zip
   creating: DATASET/Actor_01/
  inflating: DATASET/Actor_01/03-02-01-01-01-01-01.wav
  inflating: DATASET/Actor_01/03-02-01-01-01-02-01.wav
  inflating: DATASET/Actor_01/03-02-01-01-02-01-01.wav
  inflating: DATASET/Actor_01/03-02-01-01-02-02-01.wav
  inflating: DATASET/Actor_01/03-02-02-01-01-01-01.wav
  inflating: DATASET/Actor_01/03-02-02-01-01-02-01.wav
  inflating: DATASET/Actor_01/03-02-02-01-02-01-01.wav
  inflating: DATASET/Actor_01/03-02-02-01-02-02-01.wav
```

Features extraction

Feature extraction is a process in machine learning where the most relevant features or attributes are selected from raw data to be used as input for an algorithm. It involves identifying important patterns or characteristics in the data that can help improve the accuracy of the model. This process is important because it reduces the amount of data that needs to be processed and analyzed, thereby making the algorithms faster and more efficient. Feature extraction can involve techniques such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), or various statistical methods to select the most relevant features.



```
count = 0

# Extracting information from filename
for file in os.listdir(base_dir_path):
    filename = file.split('.')[0].split('-')
    if(len(filename)==7):
        path = base_dir_path + file
        src = int(filename[1])
        actor = int(filename[-1])
        emotion = int(filename[2])
        if int(actor)%2 == 0:
            gender = "female"
        else:
            gender = "male"

        if filename[3] == '01':
            intensity = 0
        else:
            intensity = 1

        if filename[4] == '01':
```



```
train_data = extract_features(df_train)
test_data = extract_features(df_test)
```

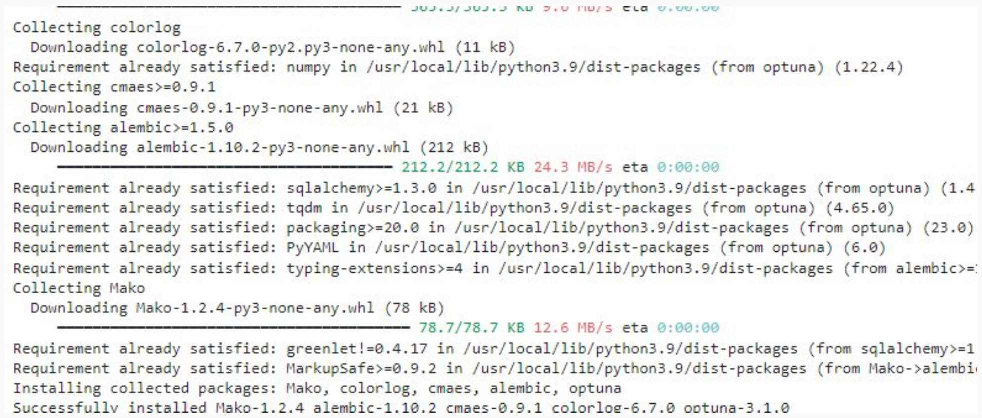
```
100%|██████████| 4201/4201 [13:08<00:00,  5.33it/s]
100%|██████████| 1051/1051 [03:09<00:00,  5.54it/s]
```

```
train_data = train_data.dropna(how='any',axis=0)
test_data = test_data.dropna(how='any',axis=0)
```

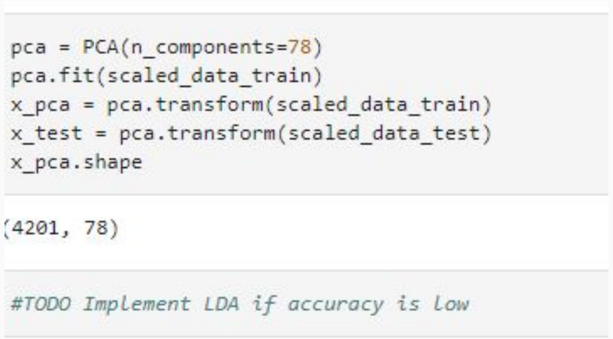
```
train_data.to_csv("train_features.csv", index=False)
test_data.to_csv("test_features.csv", index=False)
```

Dimensional reduction

Dimensionality reduction is a process of reducing the number of features or variables in a dataset while retaining the most important information. This process is useful when working with high-dimensional data because it reduces the complexity of the dataset, making it easier to analyze and visualize. In a dataframe, dimensional reduction can be achieved through techniques such as Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE). These techniques identify the most important variables in the dataset and combine them into a smaller set of variables or dimensions, without losing important information. Dimensionality reduction can help improve the accuracy and efficiency of machine learning models.



```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.3/363.3 KB 9.6 MB/s eta 0:00:00
Collecting colorlog
  Downloading colorlog-6.7.0-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from optuna) (1.22.4)
Collecting cmaes>=0.9.1
  Downloading cmaes-0.9.1-py3-none-any.whl (21 kB)
Collecting alembic>=1.5.0
  Downloading alembic-1.10.2-py3-none-any.whl (212 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.2/212.2 KB 24.3 MB/s eta 0:00:00
Requirement already satisfied: sqlalchemy>=1.3.0 in /usr/local/lib/python3.9/dist-packages (from optuna) (1.4
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from optuna) (4.65.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from optuna) (23.0)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.9/dist-packages (from optuna) (6.0)
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.9/dist-packages (from alembic>=
Collecting Mako
  Downloading Mako-1.2.4-py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.7/78.7 KB 12.6 MB/s eta 0:00:00
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.9/dist-packages (from sqlalchemy>=1
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.9/dist-packages (from Mako->alembi
Installing collected packages: Mako, colorlog, cmaes, alembic, optuna
Successfully installed Mako-1.2.4 alembic-1.10.2 cmaes-0.9.1 colorlog-6.7.0 optuna-3.1.0
```



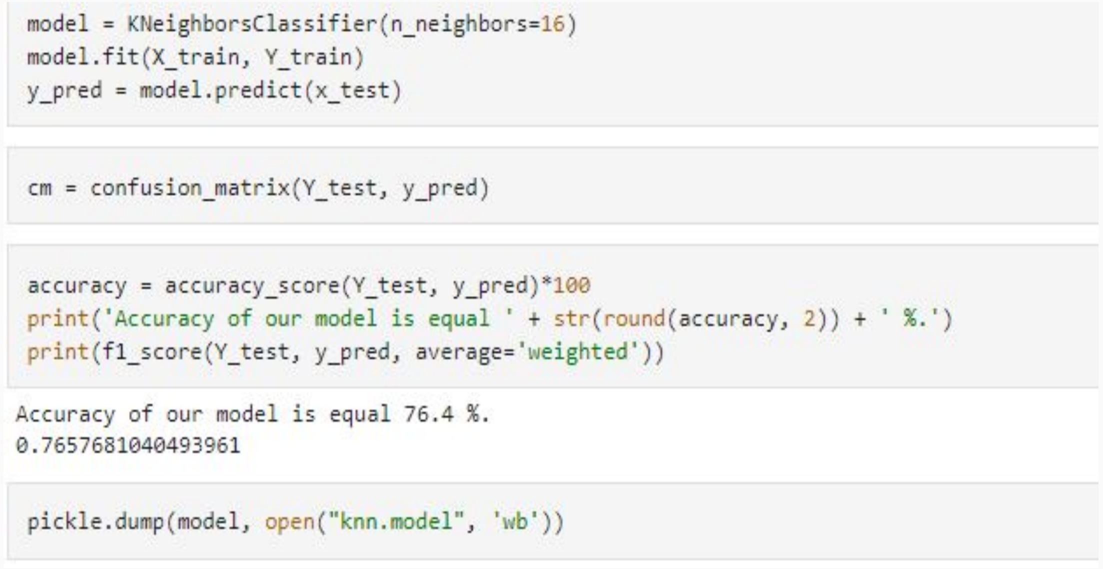
```python
pca = PCA(n_components=78)
pca.fit(scaled_data_train)
x_pca = pca.transform(scaled_data_train)
x_test = pca.transform(scaled_data_test)
x_pca.shape
```

```
(4201, 78)
```

```python
#TODO Implement LDA if accuracy is low
```

Implementing K-means Nearest Neighbour algorithm

K-means Nearest Neighbor (KNN) is a clustering algorithm that groups data points into K clusters based on their similarity. The algorithm works by selecting K initial centroids, and then iteratively assigning each data point to the nearest centroid and recalculating the centroid positions. This process continues until the centroids no longer change, indicating convergence. KNN is a simple and efficient algorithm that can be used for a variety of applications such as image segmentation and customer segmentation.



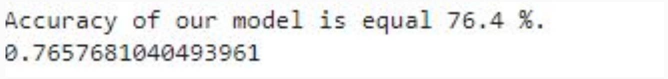
```
model = KNeighborsClassifier(n_neighbors=16)
model.fit(X_train, Y_train)
y_pred = model.predict(x_test)
```

```
cm = confusion_matrix(Y_test, y_pred)
```

```
accuracy = accuracy_score(Y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
print(f1_score(Y_test, y_pred, average='weighted'))
```

```
Accuracy of our model is equal 76.4 %.
0.7657681040493961
```

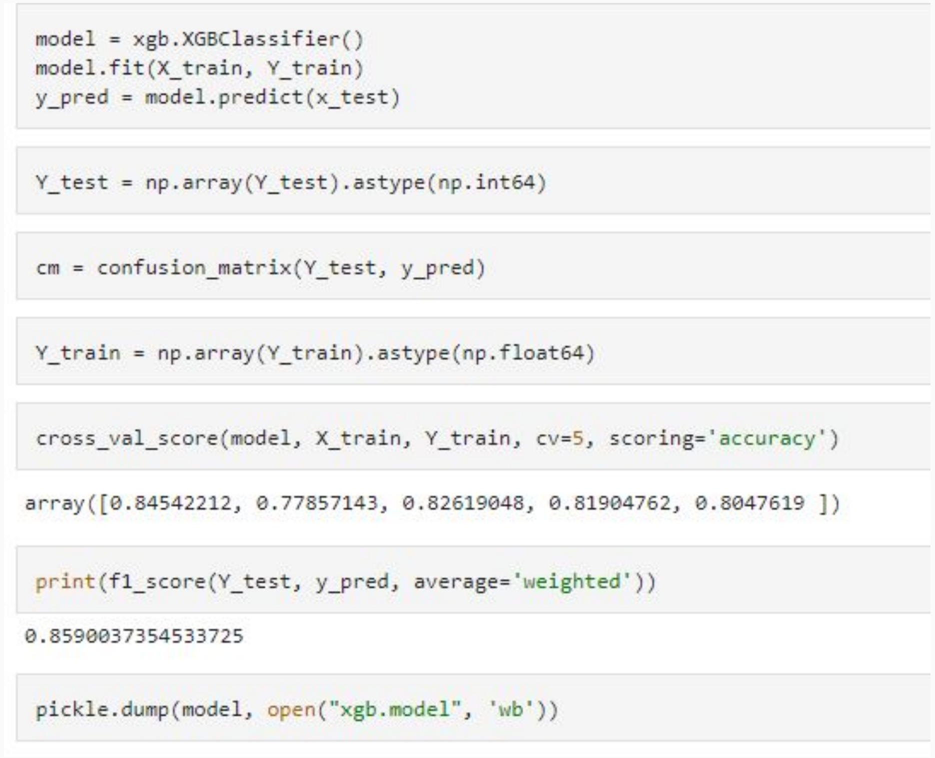
```
pickle.dump(model, open("knn.model", 'wb'))
```



```
Accuracy of our model is equal 76.4 %.
0.7657681040493961
```

Implementing XGBoost

XGBoost is a popular machine learning algorithm used for regression, classification, and ranking tasks. It is a decision-tree-based ensemble method that combines the strengths of both boosting and bagging to improve model accuracy and generalization. XGBoost implements a gradient boosting framework and uses a series of decision trees to learn from the data. It is highly scalable, efficient, and often outperforms other popular algorithms in competitions and real-world applications. XGBoost is widely used in various fields such as finance, healthcare, and natural language processing.



```
model = xgb.XGBClassifier()
model.fit(X_train, Y_train)
y_pred = model.predict(x_test)
```

```
Y_test = np.array(Y_test).astype(np.int64)
```

```
cm = confusion_matrix(Y_test, y_pred)
```

```
Y_train = np.array(Y_train).astype(np.float64)
```

```
cross_val_score(model, X_train, Y_train, cv=5, scoring='accuracy')
```

```
array([0.84542212, 0.77857143, 0.82619048, 0.81904762, 0.8047619 ])
```

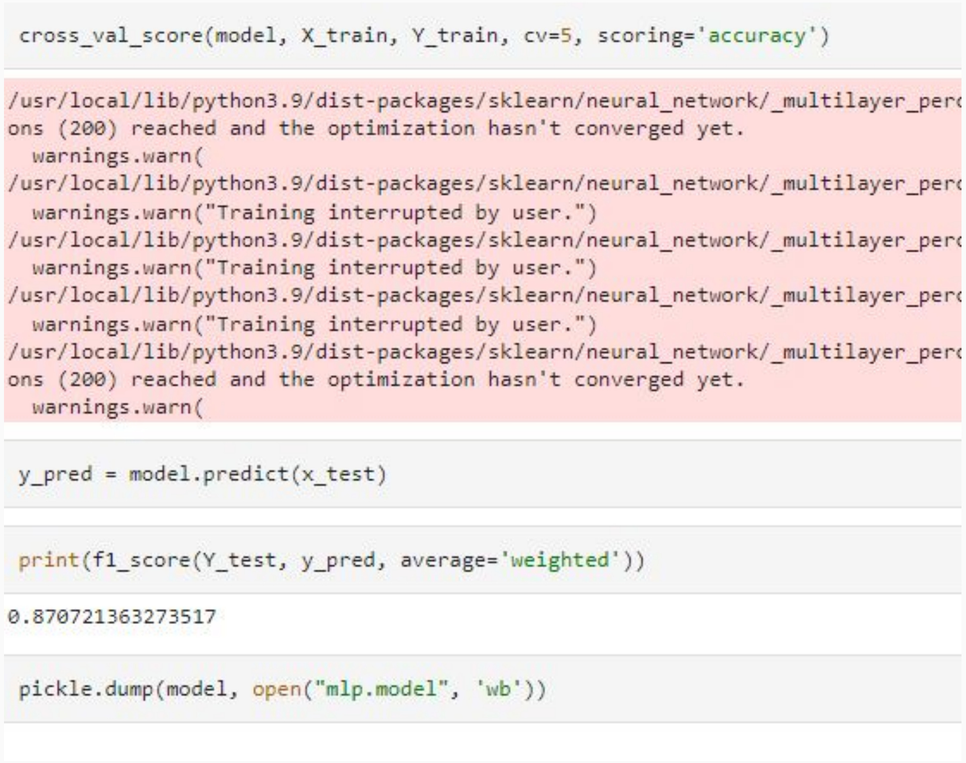
```
print(f1_score(Y_test, y_pred, average='weighted'))
```

```
0.8590037354533725
```

```
pickle.dump(model, open("xgb.model", 'wb'))
```

Implementing Multilayer Perceptron

Multilayer Perceptron (MLP) is a type of artificial neural network (ANN) used for classification, regression, and pattern recognition. It consists of multiple layers of interconnected nodes or neurons that process input data and produce output predictions. MLP uses a backpropagation algorithm to adjust the weights and biases of the neurons during training, allowing it to learn from the data and improve accuracy over time. MLPs are widely used in various fields such as computer vision, speech recognition, and natural language processing.



```
cross_val_score(model, X_train, Y_train, cv=5, scoring='accuracy')
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perc
ons (200) reached and the optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perc
  warnings.warn("Training interrupted by user.")
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perc
  warnings.warn("Training interrupted by user.")
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perc
  warnings.warn("Training interrupted by user.")
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perc
ons (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
y_pred = model.predict(x_test)
```

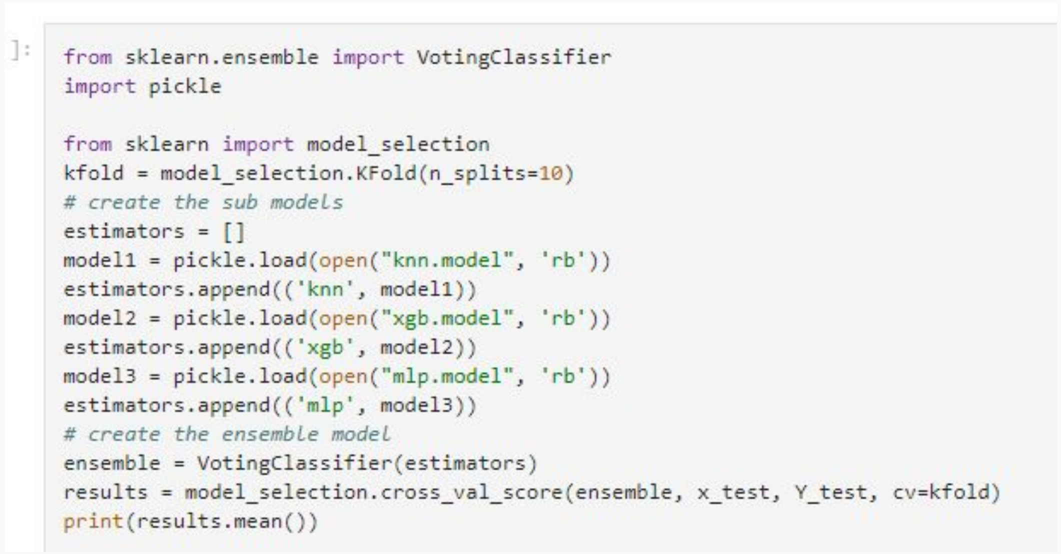
```
print(f1_score(Y_test, y_pred, average='weighted'))
```

```
0.870721363273517
```

```
pickle.dump(model, open("mlp.model", 'wb'))
```

Ensembling 3 models together

Ensembling MLP, XGBoost, and KNN into one model involves combining the predictions of each individual model to make a final prediction. This can be achieved by either averaging the predictions or using a more complex method such as stacking. Ensembling can help improve the overall accuracy and reliability of the model by leveraging the strengths of each individual algorithm. However, ensembling requires additional computational resources and careful tuning to optimize performance.



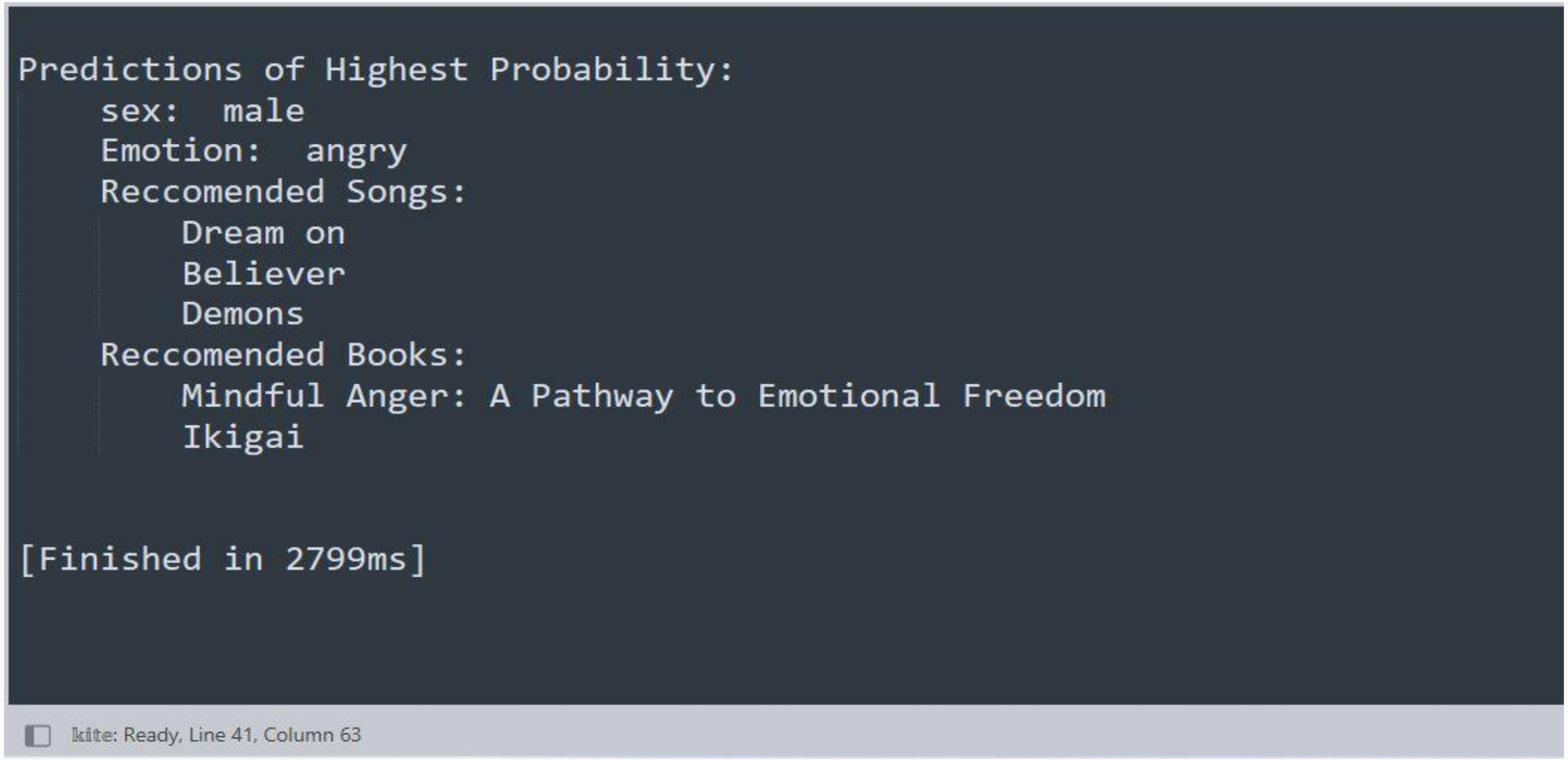
```
]:  from sklearn.ensemble import VotingClassifier
    import pickle

    from sklearn import model_selection
    kfold = model_selection.KFold(n_splits=10)
    # create the sub models
    estimators = []
    model1 = pickle.load(open("knn.model", 'rb'))
    estimators.append(('knn', model1))
    model2 = pickle.load(open("xgb.model", 'rb'))
    estimators.append(('xgb', model2))
    model3 = pickle.load(open("mlp.model", 'rb'))
    estimators.append(('mlp', model3))
    # create the ensemble model
    ensemble = VotingClassifier(estimators)
    results = model_selection.cross_val_score(ensemble, x_test, Y_test, cv=kfold)
    print(results.mean())
```

87%

using MLP

Whereas for the XGBoost - 85.9%

KNN - 76.75%



```
Predictions of Highest Probability:
    sex:  male
    Emotion:  angry
    Reccomended Songs:
        Dream on
        Believer
        Demons
    Reccomended Books:
        Mindful Anger: A Pathway to Emotional Freedom
        Ikigai


[Finished in 2799ms]
```

kite: Ready, Line 41, Column 63

Future prospects about progress

THINGS YET TO BE DONE...

We are working hard on more features and code improvements.We also plan to use it in a wide range of spectrum and other advanced training algorithms

- Trying improve the latency and accuracy of the model
- Training the module based on huge dataset
- Deploying the idea on the various platform , including desktops , laptops, websites and android and ios.
- Making the module more proficient and reliable to use

Thanks!

Contact us:

MeowTech
(will be available soon)

Contact

github.com/Priyangshu1711
github.com/debajyotisarkarhome
github.com/Dunston-Chiro
https://github.com/cytric-74