ОГЛАВЛЕНИЕ

Введение ... 3
Процесс машинного обучения ... 4
Анализ датасета RAVDESS ... 5
Конструирование признаков ... 7
Кратковременное преобразование Фурье ... 8
Мел-кепстральные коэффициенты ... 10
Мел-спектрограммы ... 11
Цветность (Chromagram) ... 11
Извлечение объектов ... 13
Исследование модели ... 15
Классические модели машинного обучения ... 16
SVC ... 18
kNN ... 19
Random Forest ... 20
Построение и обучение модели классификатора MLP ... 21
Построение модели классификатора MLP ... 21
Обучение модели классификатора MLP ... 24
Оценка производительности и изучение вариантов улучшения модели ... 25
Матрица ошибок ... 26
K-Fold кросс-валидация ... 28
Кривая проверки ... 30
Кривая обучаемости ... 31
Заключение ... 33
Список использованных источников ... 34

Введение

В современном мире активно используются голосовые сообщения для коммуникации между людьми. Мы используем данный вид связи, чтобы быстро передать необходимую информацию до нужного нам человека. Аудиофайлы используются практически везде. Мы используем их и для работы, и для развлечения. К примеру, это может быть видеоконференция по работе, в которой голоса передаются через аудиофайлы на различные устройства.

Но если посмотреть на аудиофайлы под другим взглядом, то компьютер можно научить анализировать, что предоставлено в таком файле. Это одна из активно развивающихся сфер, позволяющая человека взаимодействовать с техническим устройством использую голосовые команды. Существует множество голосовых помощников, которые нацелены сделать жизнь людей легче. Есть люди, которым сложно взаимодействовать с техническими устройства путём классического физического взаимодействия из-за ряда проблем со здоровьем. Им голосовое управление наиболее актуально, так как оно помогает без особых трудностей взаимодействовать с устройствами.

На основе голосового аудиофайла можно проанализировать в каком эмоциональном состоянии находится человек. Особенно актуально это в компаниях, которые работают в сфере психологической поддержки. Это позволит сразу понять ментальное состояние человека. Можно определять множество других особенностей по аудиофайлу. Всё это стало возможным с машинным обучением. Модель можно научить распознавать по различным признакам эмоции человека в данный момент, либо же ориентировочный возраст. Вариантов использования большое множество.

В данной работе будет реализовываться распознавание речевых эмоций на основе аудиофайлов с голосом людей. Исследуется наиболее распространенные модели машинного обучения, в частности те, которые доступны в готовом виде в scikit-learn. Scikit-learn – это один из наиболее часто используемых пакетов Python для машинного обучения. Он предоставляет множество алгоритмов,

которые реализованы внутри данной библиотеки. Для обработки аудиофайлов нам потребуется использовать пакет под названием librosa, который позволит проводить анализ аудиофайлов.

После сравнительного анализа производительности с моделями машинного обучения мы обучаем многослойную модель персептрона (MLP) для классификации в попытке распознать эмоцию, переданную в звуковом фрагменте речи. Классификаторы MLP - хорошая модель DNN для начала, потому что они просты, гибки и подходят, когда входным данным присваивается метка - в нашем случае эмоция.

Используем набор данных RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song dataset), созданный Стивеном Ливингстоном и Фрэнком Руссо из Университета Райерсона. Для работы берётся только речевая часть набора данных RAVDESS, доступная только для аудио, которая содержит в себе около 200 МБ данных. Аудио получено от 24 актеров (12 мужчин, 12 женщин), повторяющих два предложения с различными эмоциями и интенсивностью. Мы получаем 1440 речевых файлов (24 актера * 60 записей на каждого актера). Каждый аудиофайл был оценен человеком 10 раз по эмоциональному качеству.

Процесс машинного обучения

Данную работу можно разделить на несколько этапов. Поговорим подробнее о каждом из них:

1. Конструирование признаков:

   Необходимо обрабатывать предоставленные аудиофайлы для этого выбирать и определить свойства, которые модель будет использовать для оценки аудиофайлов. На основе этих данных далее обучается модель, поэтому важно выбрать правильных подход к преобразованию аудиофайлов в массивы чисел, обрабатываемые в последствии.
2. Извлечение объектов:

Их каждого предоставленного нам аудиофайла необходимо вычислить информацию и создать матрицу, которую мы будем использовать для модели машинного обучения. Далее подробнее поговорим о том, что из себя представляют необходимые нам данные.

3. Исследование модели:

   Тестирование разных моделей машинного обучения на нашем наборе данных. Необходимо удостовериться, что мы выбрали правильную модели для нашей работы. Рассмотрим различные модели, которые имеют смысл для нашего набора данных.

4. Построение и обучение модели классификатора MLP:

   Выбрать и оптимизировать свойства нашей модели на основе полученных данных используя различные гиперпараметры и архитектуры.

5. Оценка производительности модели:

   Необходимо изучить качество реализованный модели. Оценить её точность по данным для обучения и сопоставить её с тестовыми данными, которые модель не видела при обучении.

6. Изучение вариантов улучшения модели:

   Проанализировать возможности улучшения качества модели машинного обучения. Проверить, подходит ли размер набора данные для модели, а также не является ли модель слишком сложной или простой

Выделены основные этапы данной работы, по которым будет реализовываться модель распознавания речевых эмоций на основе голосового аудиофайла. Работа выполнена в jupyter notebook на языке python с использованием необходимых пакетов машинного обучения и анализа аудиофайлов.

Анализ датасета RAVDESS

RAVDESS — это проверенная мультимодальная база данных эмоциональной речи и песни. База данных гендерно сбалансирована и состоит из 24 профессиональных актеров, произносящих лексически соответствующие

утверждения с нейтральным североамериканским акцентом. Датасет содержит в себе различные данные: Видеозаписи и звуковые файлы. Нас буду интересовать звуковые файлы, которые предоставлены в двух вариантах: Запись речи или запись песни, исполняемой человеком. В данной работе используются аудиофайлы с записью речи 24 актёров.

Речь включает в себя различный эмоциональный окрас:

- Нейтральное состояние (neutral)
- Спокойствие (calm)
- Радость (happy)
- Грусть (sad)
- Гнев (angry)
- Страх (fearful)
- Отвращение (disgust)
- Удивление (surprised)

Речевой файл (Audio_Speech_Actors_01-24.zip, 215 МБ) содержит 1440 файлов формата wav: 60 попыток на актера x 24 актера = 1440.

Каждый из файлов RAVDESS имеет уникальное имя. Имя файла состоит из 7-частного числового идентификатора (например, 03-01-06-01-02-01-12.wav). Эти идентификаторы определяют характеристики стимула:

Идентификаторы имени файла

1. Модальность (01 = полный AV, 02 = только видео, 03 = только аудио).
2. Вокальный канал (01 = речь, 02 = песня).
3. Эмоции (01 = нейтральный, 02 = спокойный, 03 = счастливый, 04 = грустный, 05 = злой, 06 = испуганный, 07 = отвращение, 08 = удивленный).
4. Эмоциональная интенсивность (01 = нормальная, 02 = сильная). Нейтральная» эмоция не имеет сильной интенсивности.

5. Утверждение (01 = «Дети разговаривают у двери», 02 = «Собаки сидят у двери»).
6. Повторение (01 = 1-е повторение, 02 = 2-е повторение).
7. Актер (от 01 до 24. Актеры с нечетными номерами - мужчины, актеры с четными номерами - женщины).

Конструирование признаков

Прежде чем строить какие-либо прогностические модели, нам нужны не только данные, но и пригодное для использования представление данных. Необходимо определить свойства, которые модель будет использовать для обучения.

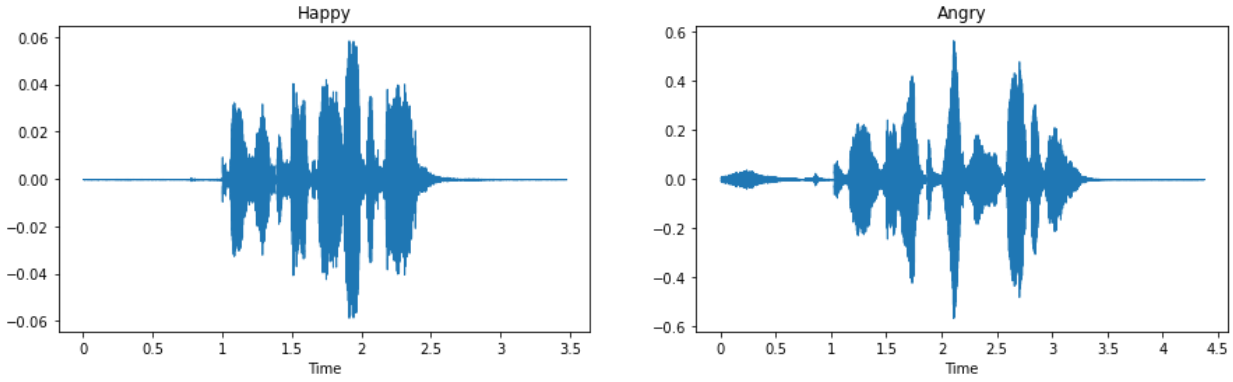
Аудиофайл может быть представлен в виде временного ряда с зависимой осью, представляющей собой амплитуду аудиосигнала. Форма сигнала звукового файла — это вся информация, которой мы располагаем для создания функций для обучения нашей модели. Однако форма сигнала не несет достаточной различающей информации, поэтому нам нужно преобразовать форму сигнала в более удобную форму.

Иногда наш входной набор данных содержит достаточно информации, чтобы обучить модель для получения точных результатов, но даже тогда входные данные должны быть тщательно изучены и преобразованы — это позволяет нам выбрать наилучшую модель для наших функций, которая обычно намного лучше и отличается от модели, которая работает с нашими исходными входными данными. Правильное проектирование функций несет в себе преимущества:

- Более точные, обобщаемые модели
- Понимание поведения модели при принятии решений
- Гибкость в выборе моделей
- Более быстрое обучение

Перед разработкой функций мы всегда очищаем наши данные, что может заключаться в удалении выбросов, удалении нерелевантных входных данных в соответствии с бизнес-логикой или удалении шума. Для аудио очистка данных может состоять в том, чтобы сделать звуковые сэмплы равными по длине и дополнить тишиной на обоих концах. Набор данных RAVDESS сделал это за нас, но это редкий случай.

Посмотрим, с чего мы начнем, сравнив аудиофайлы с эмоциями "Счастлив (Happy)" и "Зол (Angry)" Актера 1:



Хорошо, есть видимая разница, но ее недостаточно, чтобы классифицировать по эмоциям. Однако необработанные сигналы могут быть использованы для других задач классификации звука при правильном подходе.

Кратковременное преобразование Фурье

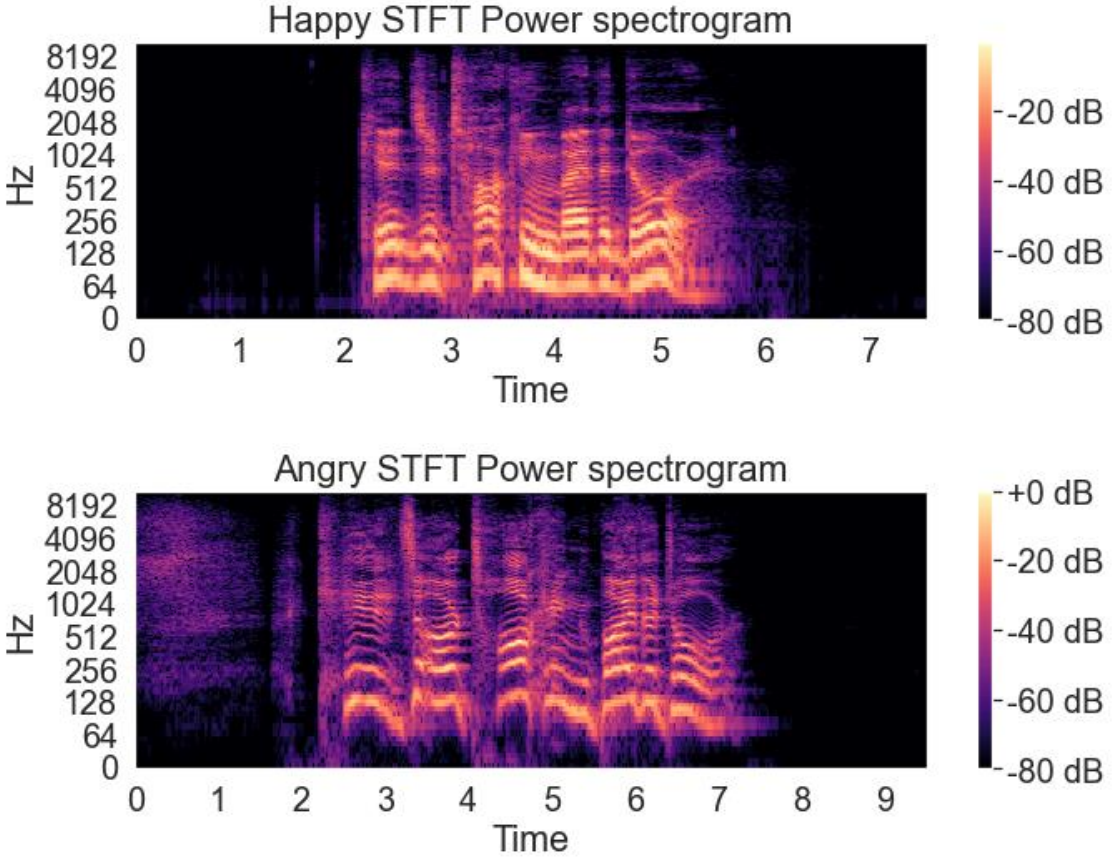
Самым низко висящим плодом в обработке сигналов временных рядов является, конечно же, преобразование Фурье. В частности, мы будем использовать Кратковременное преобразование Фурье (STFT), которое разрежет наш звуковой сигнал на короткие, перекрывающиеся сегменты одинаковой длины и использует преобразование Фурье каждого сегмента по отдельности для получения нескольких спектрограмм мощности, идентифицирующих резонансные частоты, присутствующие в нашем аудиофайле. Основным преимуществом STFT является лучшее разрешение изменений в звуковом сигнале во времени.

Для непрерывного временного сигнала x(t) кратковременное преобразование Фурье (STFT) можно записать так:

$$STFT_x\,(\omega,\tau)\;=\;\int_{-\infty}^{\infty}\mathrm{x(t)w(t}\;-\;\tau)\;\mathrm{e}^{-\mathrm{j}\omega\mathrm{t}}\;\mathrm{dt},$$

где w(t) – временная оконная функция, имеющая конечную длительность T. Умножение исходного сигнала на $\mathrm{w(t}\;-\;\tau)$ локализует интеграл Фурье в окрестности $\mathrm{t}\;-\;\tau$.

Визуализируем аудиофайлы, чтобы лучше с ними справиться. Загрузим образец аудиофайла из нашего набора данных - каждый аудиофайл длится 3 секунды.

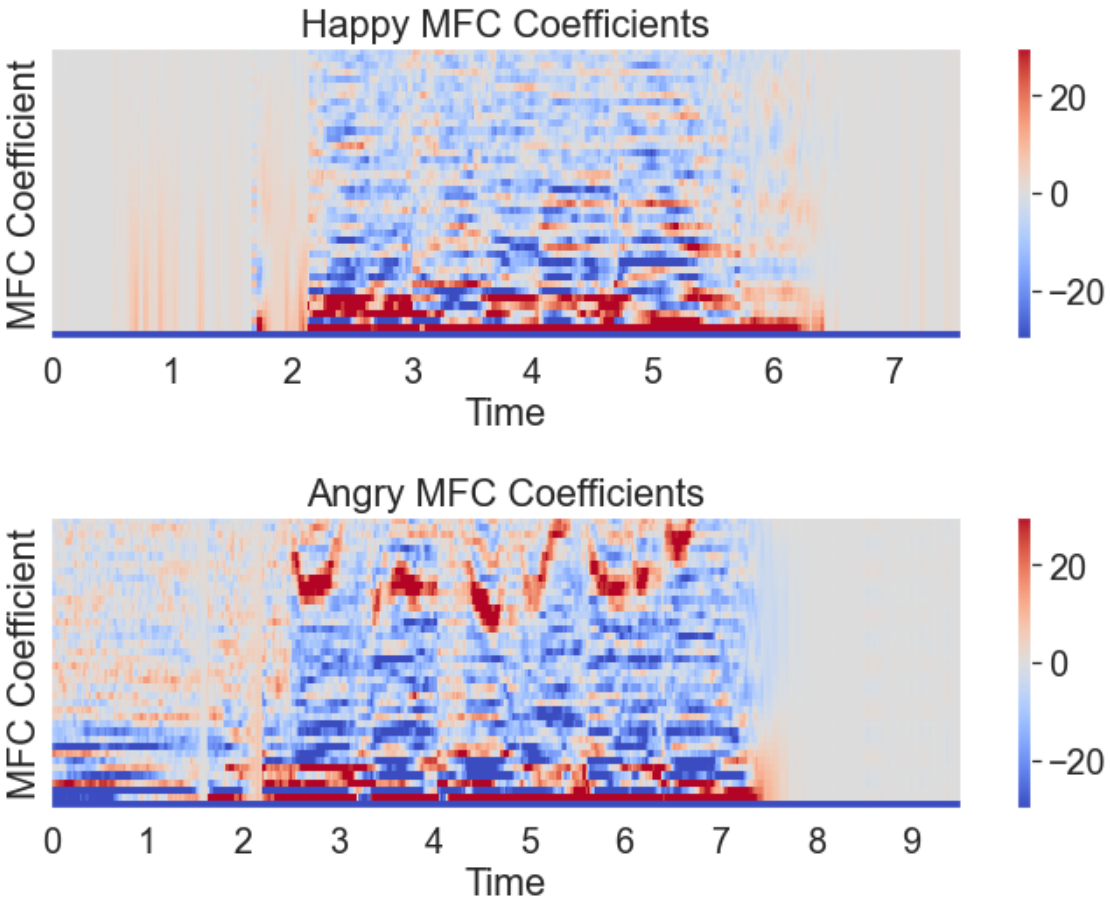


Преобразования в STFT в дальнейшем предоставят нам часть данных для обучения модели. Поскольку у нас есть 3-секундный аудиофайл, спектрограммы STFT для этих аудиофайлов объединяются, чтобы показать изменение частот в 3-секундном окне.

Мел-кепстральные коэффициенты

Мы собираемся опираться на STFT, используя в качестве функции Мел-кепстральные коэффициенты (MFCC). Вкратце, MFCC — это математический метод, который преобразует спектр мощности аудиосигнала в определенное количество коэффициентов, представляющих мощность аудиосигнала в частотной области (области высоты тона), взятой во времени.

Другими словами, коэффициенты MFC дают нам представление об изменении высоты тона аудиосигнала. Получим эти данные на основе двух аудиофайлов нашего датасета:



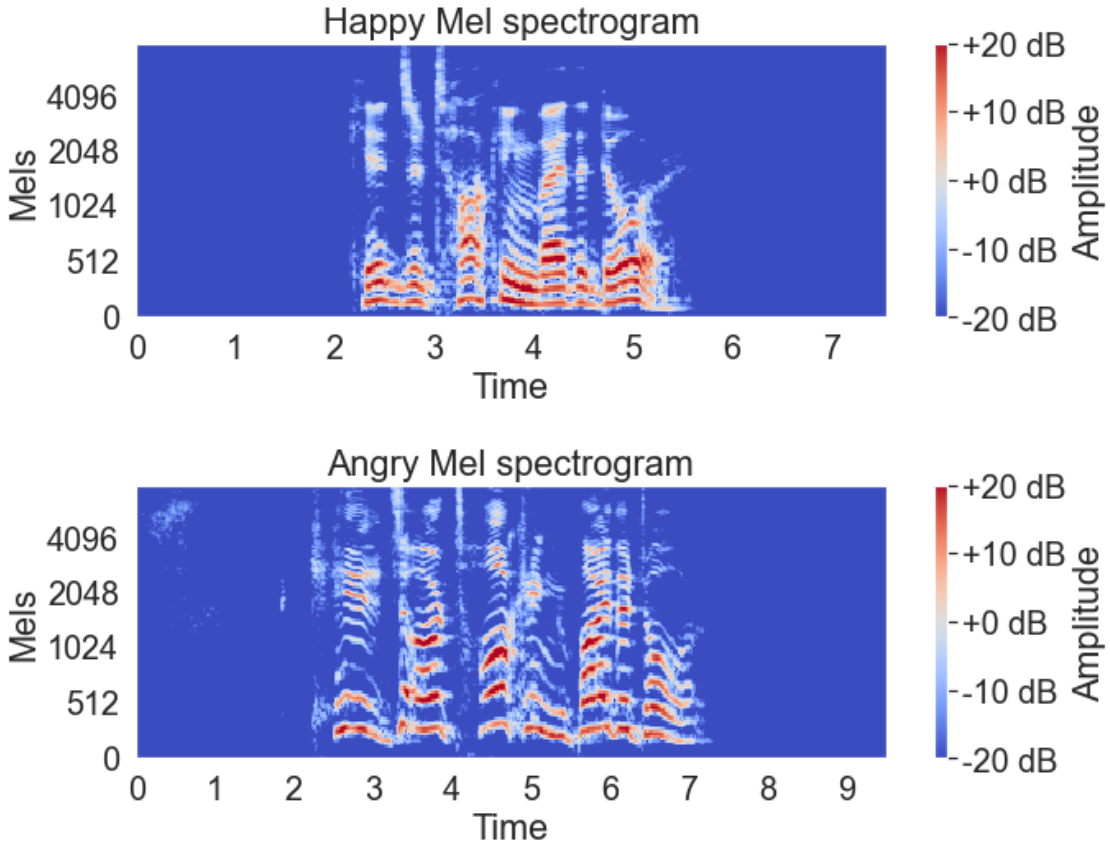
Положительные MFCC соответствуют низкочастотным областям кепстра (косинус-преобразование Фурье логарифма спектра мощности), а отрицательные эффекты - высокочастотным.

Мы видим, что злой голос имеет гораздо большую долю положительных МФФК, что соответствует более низкой высоте голоса по сравнению со

счастливым голосом. Значит, что счастливый голос имеет более легкий тон и более высокую высоту тона.

Мел-спектрограммы

Когда мы сопоставили частоты спектрограммы мощности со шкалой мэл, мы получили Мел-спектрограмму - простой аналог спектрограммы мощности со шкалой частот в мэл.



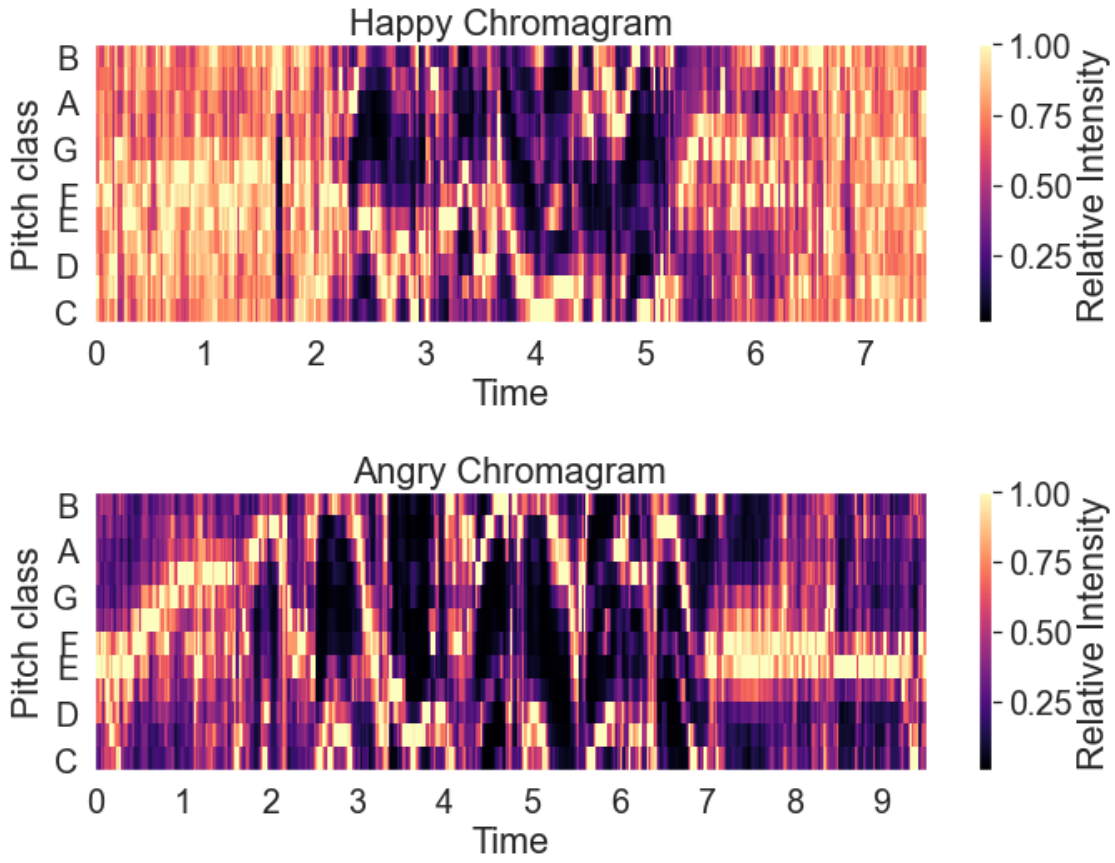
Злой голос производит более резкие переходы между пиками частоты мел по сравнению со радостным голосом, который имеет более плавные переходы высоты тона. Злой голос имеет резкие переходы, в то время как счастливый голос более ровный и приятный для слуха. Мел-спектрограммы мы будем использовать для преобразования наших данных в будущем.

Цветность (Chromagram)

Чтобы извлечь дополнительную функцию, мы собираемся построить хромаграмму на каждой спектрограмме мощности, вычисленной с помощью

STFT. Chromagram - это представление аудиосигнала во времени, сопоставляющее аудиосигнал с классом высоты тона.

Чаще всего мы сопоставляем 12 стандартных классов высоты тона (т.е. музыкальная шкала CDEFGAB + 5 полутонов дает нам 12 классов высоты тона).



Распределение высоты тона агрессивного голоса имеет гораздо меньшую дисперсию по сравнению со радостным голосом, высота тона которого имеет более высокую дисперсию в любой момент времени. Тон злого голоса более интенсивный и "отрывистый" по сравнению со радостным голосом, который более мягкий для ушей.

Отобраны признаки, по которым будет реализована модель машинного обучения, а именно: Цветность (Chromagram), Мел-кепстральные коэффициенты, Мел-спектрограмма.

Извлечение объектов

Используем библиотеку librosa, позволяющей анализировать аудио. Librosa абстрагирует всю математику от нашей работы.

Поскольку Хромаграмма, Мел-спектрограмма и MFCC вычисляются на аудиокадрах, созданных STFT, мы собираемся получить матрицу обратно из каждой функции, поэтому мы возьмем среднее значение этих матриц, чтобы создать единый массив признаков для каждого признака и каждого образца звука, чтобы получить 3 массива признаков на образец звука.

Хромаграмма создаст 12 объектов. По одному для каждого из 12 классов высоты тона.

Мел-спектрограмма предоставит 128 объектов; Мы определили количество полос частот мел при n_mels=128

Мел-кепстральные коэффициенты (MFCC) выдаст 40 MFCC. Установим количество возвращаемых коэффициентов на n_mfcc = 40.

Реализованы необходимые функции для получения данных для обучения моделей, преобразуем наш датасет и посмотрим на него:

Количество обработанных аудиофайлов: 1440

Количество числовых характеристик, извлеченных для каждого элемента: 180

	0	1	2	3	4	5	6	\
0	0.762871	0.786686	0.769217	0.768762	0.773605	0.766615	0.770437	
1	0.747417	0.781693	0.774763	0.755513	0.770826	0.783455	0.754270	
2	0.767132	0.788347	0.785653	0.786579	0.777644	0.755638	0.761176	
3	0.765556	0.760809	0.754814	0.773621	0.797597	0.785697	0.754200	
4	0.706621	0.751378	0.765777	0.754597	0.759112	0.770332	0.755594	
…	…	…	…	…	…	…	…	
1435	0.633462	0.580423	0.539717	0.541983	0.569329	0.576741	0.595737	
1436	0.667961	0.637937	0.599195	0.591554	0.581671	0.556208	0.566106	
1437	0.640290	0.588677	0.548105	0.532067	0.551831	0.579697	0.555725	
1438	0.612517	0.594626	0.586922	0.566910	0.584284	0.618568	0.632370	
1439	0.682341	0.621409	0.606023	0.590485	0.575120	0.597221	0.615632	

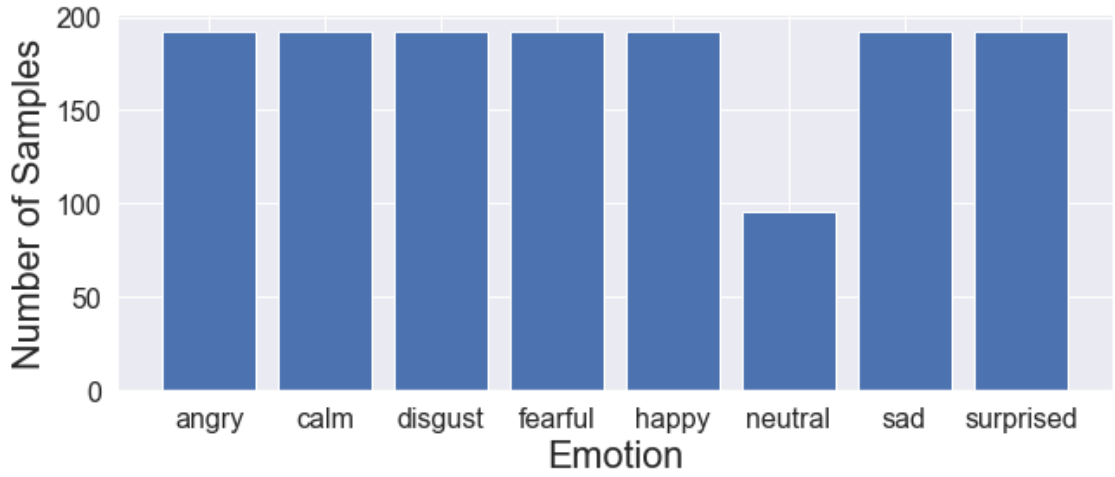
```
          7         8         9   ...       170       171       172  \
0    0.764894  0.780340  0.761150  ...  0.457082 -1.399109 -2.926856
1    0.748580  0.766922  0.768814  ...  0.275460 -2.521470 -2.987673
2    0.752333  0.774452  0.743741  ... -0.002119 -0.909152 -3.045955
3    0.761343  0.742356  0.725235  ... -0.403806 -1.329651 -2.513405
4    0.741855  0.750051  0.755684  ...  0.206463 -2.188582 -2.835501
...       ...       ...       ...  ...       ...       ...       ...
1435 0.595334  0.587991  0.582093  ... -2.279651 -0.956125  1.618874
1436 0.563750  0.556274  0.562026  ...  1.828094  2.644069  1.914414
1437 0.533518  0.550872  0.563250  ... -0.276459  0.734496  0.793300
1438 0.574342  0.554251  0.568200  ...  1.947711  3.540176  0.957979
1439 0.617779  0.630261  0.629224  ...  2.187292  1.822321  0.664088

          173       174       175       176       177       178       179
```

У нас есть матрица 1440 x 180. 1440 аудиофайлов, по одному в строке, с серией из 180 числовых признаков для каждого аудио.

Каждый из 1440 объектов содержит 180 признаков, состоящих из 12 классов высоты тона + 128 полос Мел-спектрограмм + 40 коэффициентов MFC.

Посмотрим на баланс классов в нашем наборе данных:



Отлично, классы кажутся сбалансированными. Это облегчает задачу. Все эмоции, кроме нейтрального класса, имеют "сильную" интенсивность, поэтому нейтральных образцов вдвое меньше. Это может отразиться на наших моделях.

Исследование модели

Чтобы правильно обучить большинство моделей машинного обучения на наборах данных, сначала необходимо масштабировать данные. Это имеет важное значение для моделей, которые вычисляют расстояния между данными, и особенно важно для моделей глубокого обучения: если существует разница в дисперсии объектов просто из-за их возможного диапазона значений, то модель узнает, что объекты с наибольшей дисперсией являются наиболее важными.

Посмотрим на свойства наших данных. Программную реализацию получения этих данных можно посмотреть в документах приложенных к данной работе. Сейчас же проанализируем полученное описание:

```
12 Chromagram метрики:
min = 0.310,     max = 0.874,     Среднее = 0.666,     Отклонение = 0.085

128 Mel Spectrogram метрики:
min = 0.000,     max = 149.208,    Среднее = 0.187,    Отклонение = 1.597

40 MFCC метрики:
min = -873.242,  max = 115.126,  Среднее = -14.635,   Отклонение = 98.558
```

По результатам видно, что наши признаки принадлежат к разным распределениям: отклонение наших коэффициентов MFC на порядки больше, чем у других признаков. Это не означает, что коэффициенты MFC являются наиболее важной характеристикой, скорее это свойство способа их вычисления. Потребуется масштабировать этот набор функций.

Есть выбор между стандартным и минимальным масштабированием sklearn. Стандартное масштабирование вычитает среднее значение каждого объекта и делит его на стандартное отклонение этого объекта, создавая объекты со средним значением, равным нулю, и единичной дисперсией, то есть дисперсией и стандартным отклонением, равными 1. Минимальное-максимальное масштабирование преобразует каждый объект в пределах указанного ограниченного интервала.

Преобразуем наши признаки:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

scaler = StandardScaler()
# Стандартное масштабирование
data_scaled = data
data_scaled = scaler.fit_transform(data_scaled)

scaler = MinMaxScaler()
# Мин/Макс масштабирование
data_minmax = data
data_minmax = scaler.fit_transform(data_minmax)
```

```
Стандартное преобразование:

12 Chromagram метрики:
min = -3.949,    max = 2.640,     Среднее = 0.000,     Отклонение = 1.000

128 Mel Spectrogram метрики:
min = -0.476,    max = 36.543,    Среднее = -0.000,    Отклонение = 1.000

40 MFCC метрики:
min = -4.208,    max = 6.243,    Среднее = -0.000,    Отклонение = 1.000


Мин\Макс преобразование:

12 Chromagram метрики:
min = 0.000,     max = 1.000,     Среднее = 0.611,     Отклонение = 0.181

128 Mel Spectrogram метрики:
min = 0.000,     max = 1.000,     Среднее = 0.015,     Отклонение = 0.060

40 MFCC метрики:
min = 0.000,    max = 1.000,    Среднее = 0.406,    Отклонение = 0.166
```

Нулевое среднее значение и единичная дисперсия для стандартного масштабирования и в диапазоне [0,1] для минимального максимального масштабирования - значение по умолчанию, когда мы не указываем значения.

Классические модели машинного обучения

Время, необходимое для адаптации классических моделей машинного обучения, часто ограничено размером набора данных. Машина опорных векторов (SVM) является ярким примером, поскольку она ограничена своей квадратичной сложностью без учета количества выборок. Напротив, подгонка даже самого простого DNN - MLP - сопряжена со значительно более высокой

временной сложностью: в то время как MLP является линейным с учетом выборок, он ограничен экспоненциальной временной сложностью с учетом количества скрытых слоев из-за использования обратного распространения. В случае 1 скрытого слоя - что, как мы увидим, бывает достаточно часто - MLP конкурирует с некоторыми классическими моделями по временной сложности.

Кроме того, можно привести веские аргументы в пользу использования классического метода машинного обучения из-за небольшого размера нашего набора данных; Некоторые из наиболее надежных моделей, таких как классификаторы опорных векторов (машинные) (SVC) и классификаторы k-ближайших соседей (kNN), особенно подходят для небольших наборов данных и падают отдельно от огромных наборов данных.

Чтобы сравнить модели, нам придется оценить их производительность. Самый простой способ сделать это - обучить модель на части нашего набора данных и протестировать ее на оставшейся части. Мы будем использовать train_test_split от sklearn для создания стандартного разделения теста 80/20. Модель подходит для 80% данных и протестирована на производительность по сравнению с 20% данных, чего она никогда не видела в обучении

Подобным образом разделяем масштабированные данные с помощью двух различных преобразований, описанных ранее.

Мы попробуем каждую готовую модель машинного обучения от sklearn и выберем несколько для изучения, поскольку эти модели будут обучаться практически мгновенно на этом наборе данных.

```
# Модели для анализа
classification_models = [
    KNeighborsClassifier(),
    SVC(kernel='linear'),
    SVC(kernel='rbf'),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]
```

```
scores = []
for model in classification_models:
    model.fit(X_train_scaled, y_train)
    score = model.score(X_test_scaled, y_test)
    model_name = type(model).__name__
    if model_name=='SVC' and model.kernel=='rbf': model_name+=' RBF kernel'
    scores.append((model_name,(f'{100*score:.2f}%')))
# Визуализируем данные
scores_df = pd.DataFrame(scores,columns=['Classifier','Accuracy Score'])
scores_df.sort_values(by='Accuracy Score',axis=0,ascending=False)
```

	Classifier	Accuracy Score
1	SVC	57.29%
4	RandomForestClassifier	53.47%
0	KNeighborsClassifier	53.12%
2	SVC RBF kernel	47.57%
3	DecisionTreeClassifier	38.54%
5	AdaBoostClassifier	29.51%
6	GaussianNB	25.69%
7	QuadraticDiscriminantAnalysis	19.44%

Давайте выберем три лучших – Random Forest, SVC и kNN - и подробнее рассмотрим каждый из них.

SVC

Идея SVMS, на которой основана модель SVC, состоит в том, чтобы найти разделяющую гиперплоскость - подпространство с размерностью на единицу меньше, чем у пространства объектов для точек в нашем пространстве объектов. Эта идея распространяется на n измерений. Если точки разделимы гиперплоскостью, то они называются линейно разделимыми. Поскольку существует бесконечное количество возможных разделяющих гиперплоскостей для любого линейно разделяемого пространства объектов, SVM вычисляет, какие точки находятся ближе всего к каждой такой гиперплоскости, и использует их для построения опорного вектора. SVM выбирает гиперплоскость, которая максимизирует расстояние - запас - до каждого опорного вектора. Таким образом, мы максимизируем разделяющую способность выбранной гиперплоскости.

```
model = SVC(
    C=10,
    gamma='auto',
    kernel='rbf',
```

```
    random_state=69
)

model.fit(X_train, y_train)


SVC Model's accuracy on training: 100.00%
SVC Model's accuracy on test: 52.43%
```

Совсем неплохо для относительно простой модели SVC. C - параметр регуляризации. Возможно, было бы неплохо еще больше оптимизировать модель SVC, если мы не найдем лучшую. В нынешнем виде мы стремимся к значительно более высокой производительности в этой задаче.

kNN

kNN - следующий на очереди, проверенный метод машинного обучения 70-х годов. kNN имеет большой интуитивный смысл: представьте, что вы наносите точки на график и рисуете ворота вокруг точек, которые выглядят так, как будто они принадлежат к одной и той же группе. Вот что это такое - мы строим объекты наших обучающих выборок и сравниваем расстояние объектов тестовой выборки до всех этих точек; затем просто берем k ближайших точек к тестовой выборке и выбираем наиболее часто встречающуюся метку / класс.

```
######## Default kNN  ########
model = KNeighborsClassifier(
)

model.fit(X_train, y_train)

##### tuned kNN ######
model = KNeighborsClassifier(
    n_neighbors = 5,
    weights = 'distance',
    algorithm = 'brute',
    leaf_size = '30',
    n_jobs=4
)

model.fit(X_train, y_train)

Default kNN Model's accuracy on training: 66.84%
Default kNN Model's accuracy on test: 43.75%

kNN Model's accuracy on training: 100.00%
kNN Model's accuracy on test: 49.65%
```

Рассмотрим следующие модели, данной точности нам недостаточно.

Random Forest

Random Forest - модель из 21-го века (2001). Мы обучаем множество различных деревьев решений, которые, по сути, представляют собой направленные ациклические графы (DAG), несколько похожие на блок-схему.

Random Forest являются отличными моделями для использования в качестве эталона из-за их низкой трудоемкости обучения и их устойчивости к неизвестным распределениям и выбросам в наборе данных, что означает, что Random Forest требуют относительно небольшого исследовательского анализа данных.

```
####### Default Random Forest ########
model = RandomForestClassifier(
    random_state=69
)

model.fit(X_train, y_train)

########## Tuned Random Forest #######
model = RandomForestClassifier(
    n_estimators = 500,
    criterion ='entropy',
    warm_start = True,
    max_features = 'sqrt',
    oob_score = 'True',
    random_state=69
)

model.fit(X_train, y_train)
```

```
Default Random Forest Model's accuracy on training set is 100.00%
Default Random Forest Model's accuracy on test set is 52.43%

Random Forest Model's accuracy on training set is 100.00%
Random Forest Model's accuracy on test set is 56.60%
```

Неплохо для нулевых усилий, вложенных в модель по умолчанию. Random Forest являются хорошей эталонной моделью, особенно когда они ограничены во времени. Для сравнения, SVC с гиперпараметрами по умолчанию выдает ничтожную точность в 30% для этой задачи.

Построение и обучение модели классификатора MLP

Построение модели классификатора MLP

Сначала мы собираемся попробовать классификатор многослойного персептрона (MLP), простую модель искусственной нейронной сети (ANN), хорошо подходящую для предсказаний, обученных на помеченных входных данных. Обратите внимание, что модель MLP также может быть обучена регрессии. Сеть MLP состоит из входного уровня, n скрытых слоев и выходного уровня.

Мы инициализируем классификационную модель MLP со случайными весами и смещениями, равными нулю, что является стандартным способом. Мы попробуем готовую модель MLP, которая поставляется вместе со sklearn.

```
# MLP from sklearn
model = MLPClassifier(
    random_state = 69
)

model.fit(X_train, y_train)
```

```
Emotions on model:['angry' 'calm' 'disgust' 'fearful' 'happy' 'neutral'
'sad' 'surprised']
Unscaled MLP Model's accuracy on training: 65.28%
Unscaled MLP Model's accuracy on test set: 51.74%
```

Посмотрим, какие результаты получим на преобразованных данных:

```
MinMax scaled MLP Model's accuracy on training: 64.32%
MinMax sacled MLP Model's accuracy on test: 52.43%

Standard scaled MLP Model's accuracy on training: 99.74%
Standard scaled MLP Model's accuracy on test: 64.93%
```

В данном случае стандартное преобразование предоставляет наибольшую точность, которая нам и необходима.

Перед обучением сети мы должны выбрать гиперпараметры, которые определяют поведение сети при обучении.

Модель MLP имеет ряд гиперпараметров, важных для ее поведения в процессе обучения:

- Alpha: Ограничивает веса модели в пределах определенной границы для решения проблемы переобучения в диапазоне [0,1]
- Activation function: Определяет выходной сигнал нейрона с помощью преобразования, применяемого к набору входных сигналов для этого нейрона.
- Solver: Алгоритм, используемый для оптимизации наших весов (с обратным градиентным спуском в случае классификатора MLP)
- Learning rate: Насколько велико изменение, вносимое алгоритмом оптимизации в веса модели на каждой итерации обучения
- Epsilon: Уникальный для решателя "adam", численная стабильность - чтобы избежать деления на ноль.

Воспользуемся преимуществами алгоритма перекрестной проверки поиска по сетке, чтобы найти наилучшие гиперпараметры для данной модели

```
# Создаём сетку гиперпараметров
parameter_space = {
    'hidden_layer_sizes': [(8,), (180,), (300,),(100,50,),(10,10,10)],
    'activation': ['tanh','relu', 'logistic'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001, 0.01],
    'epsilon': [1e-08, 0.1 ],
    'learning_rate': ['adaptive', 'constant']
}

# Создаём объект поиска по сетке
grid = GridSearchCV(
    model,
    parameter_space,
    cv=10,
    n_jobs=4)

grid.fit(X_train, y_train)
```

В результате работы кода мы определили оптимальные гиперпараметры, которые мы будем использовать для обучения нашей модели:

```
Best parameters found:
 {'activation': 'relu', 'alpha': 0.001, 'epsilon': 1e-08,
  'hidden_layer_sizes': (300,), 'learning_rate': 'adaptive',
  'solver': 'adam'}
```

Relu Activation Function

ReLU — это кусочно-линейная функция, которая будет выводить входные данные напрямую, если они положительны, в противном случае она будет выводить ноль . Она стала функцией активации по умолчанию для многих типов нейронных сетей, потому что модель, использующая ее, легче обучается и часто обеспечивает более высокую производительность.

Алгоритм оптимизации Adam

Выбран алгоритм оптимизации Adam, вариант стохастического градиентного спуска (SGD). В отличие от SGD, который поддерживает постоянную скорость обучения на протяжении каждой итерации обучения, Adam фактически "адаптирует" или изменяет скорость обучения, принимая во внимание скользящие средние первого и второго моментов (среднее значение и дисперсию) градиента на каждой итерации обучения. Имеет смысл, что grid search выбрал более сложный алгоритм - обратите внимание, однако, что он не обязательно лучше SGD для всех задач.

Адам: Alpha, Epsilon, beta_1, beta_2

Значения для alpha и epsilon - это то, что мы обычно находим стандартными во многих других задачах классификации с использованием алгоритма оптимизации Adam; alpha =0.001 и epsilon =1e-8 - это значения по умолчанию золотого стандарта. Нам также нужно будет указать beta_1 и beta_2 для Adam которые являются константами затухания для скользящих средних первого и второго моментов градиента. Мы примем стандартные значения beta_1 = 0,9, beta_2 = 0,999.

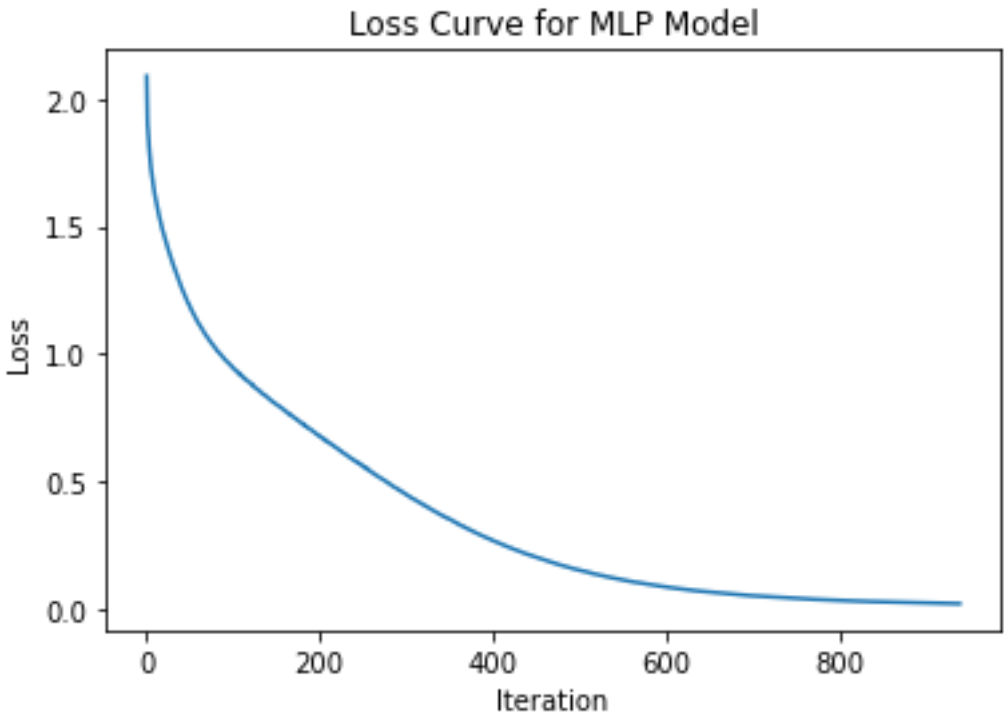
Обучение модели классификатора MLP

Посмотрим модель на основе оптимальных гиперпараметров и проанализируем её.

```
model = MLPClassifier(
    activation='relu',
    solver='adam',
    alpha=0.001,
    beta_1=0.9,
    beta_2=0.999,
    batch_size=256,
    epsilon=1e-08,
    hidden_layer_sizes=(300,),
    learning_rate='adaptive',
    max_iter=300,
    random_state=69,
)
```

```
MLP Model's accuracy on training set is 100.00%
MLP Model's accuracy on test set is 68.40%
```

Получены лучшие результаты, но этого недостаточно. Это ожидаемо, потому что большинство гиперпараметров, выбранных поиском по сетке, по умолчанию используются в MLP sklearn. Во всяком случае, наша модель MLP перенасыщает обучающие данные и плохо обобщается на тестовый набор. С почти идеальной точностью обучения ясно, что функция потерь нашей модели сходится - давайте посмотрим:



Оценка производительности и изучение вариантов улучшения модели
Потери действительно сходятся, и это, по-видимому, хорошая скорость обучения - мы хотим избежать слишком высокой скорости обучения, поскольку модель будет смещать данные в сторону последних пакетов, прошедших через нее, и избегать слишком медленной скорости обучения, потому что нашей модели потребуется слишком много времени для сходимости. Из-за его идеальной производительности на обучающих данных и низкой производительности на тестовых данных мы на данный момент подозреваем, что наша модель имеет слишком высокую дисперсию: она учится так точно подгонять свои веса к обучающим данным, чтобы получить хорошие оценки, что ее производительность не выходит за рамки этих обучающих данных. Если бы мы увидели низкую производительность как на обучающих, так и на тестовых наборах, мы бы заподозрили, что наша модель имеет высокую предвзятость.

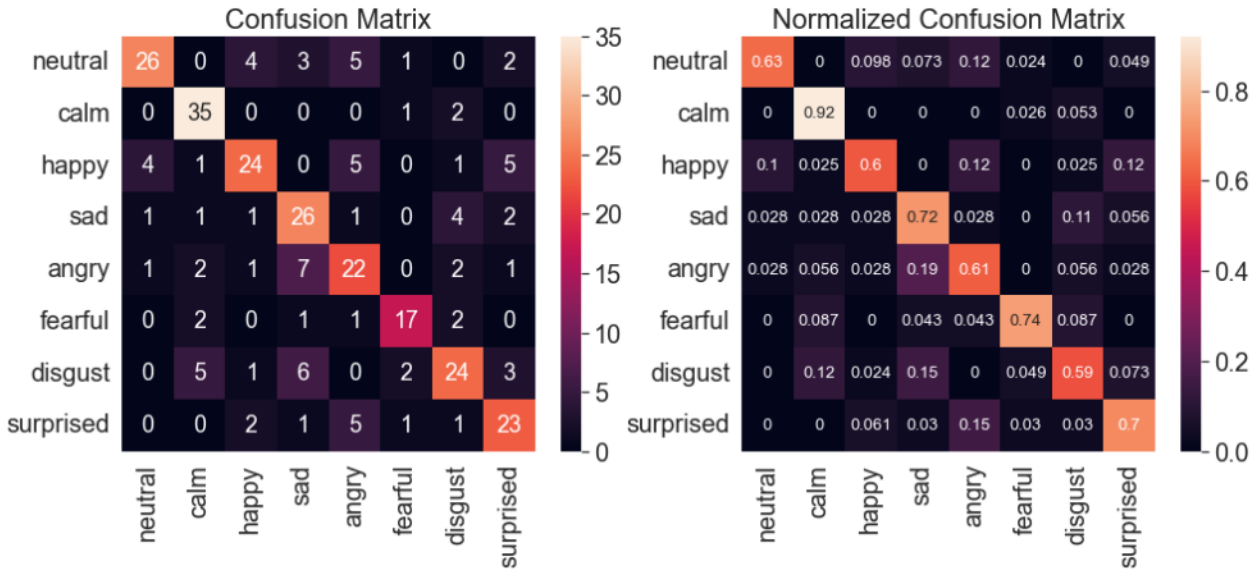
Чтобы решить проблему высокой дисперсии, при которой наша модель чрезмерно адаптируется к шуму, присутствующему в обучающих данных, мы можем уменьшить количество входных объектов и/или увеличить размер

обучающего набора, чтобы наша модель имела больше данных для изучения и могла лучше обобщать данные, которые она не видела. Мы также можем использовать регуляризацию данных, используя методы увеличения данных, такие как добавление случайного шума к звуковым выборкам.

Чтобы решить проблему высокой предвзятости, при которой наша модель недостаточно соответствует данным, мы могли бы увеличить количество входных функций, чтобы она могла лучше соответствовать базовым данным.

Матрица ошибок

Матрица ошибок описывает производительность модели классификации на тестовых данных. Оси матрицы помечены всеми возможными классами, в нашем случае эмоциями - вертикальная ось представляет предсказанные классы, в то время как горизонтальная ось представляет обрабатываемые объекты. Пересечение эмоций на диагонали матрицы — это правильно предсказанные объекты. Все недиагональные элементы являются неверными предсказаниями. Мы буквально проверяем, где наша модель сбита с толку, то есть делаем неправильные прогнозы:



Сумма элементов в левой матрице - это размер нашего тестового набора, который равен длине y_test. Это должно быть 0,2*1440 = 288. Матрица справа показывает каждый элемент в процентах от выборки в этом классе. Каждая

строка складывается до 100%, то есть каждая строка представляет все образцы определенной эмоции.

Мы можем сказать, что модель наиболее точна в предсказании «спокойствия» и наименее точна в предсказании "страха" - последнее, возможно, потому что «страх» путают со «спокойствием» и «отвращением». Эта модель также особенно борется с "нейтральным" классом, поскольку ее путают с рядом несвязанных эмоций. Основываясь на этом, мы могли бы изучить наши функции, чтобы увидеть, где они недостаточно различаются между запутанными классами, и можем ли мы переосмыслить нашу разработку функций.

Матрица ошибок — это интуитивная мера точности и отзывчивости нашей модели для каждого класса. Точность (precision) — это показатель того, сколько положительных прогнозов являются истинно положительными, а отзыв (recall) — это показатель того, сколько положительных результатов мы фактически предсказали из всех положительных выборок в наборе данных: более низкая точность означает, что у нас больше ложноположительных результатов, в то время как более низкий отзыв означает, что у нас больше ложноотрицательных результатов. Для этого набора данных точность каждой эмоции — это то, как часто мы правильно ее предсказываем, а воспоминание — это то, сколько из этих эмоций мы предсказали из всех выборок с этой эмоцией в наборе данных.

Некоторые задачи больше связаны с максимизацией точности - минимизацией ложных срабатываний, - например, с прогнозированием вероятности повторного правонарушения заключенного в случае условно-досрочного освобождения. Отказ заключенному в условно-досрочном освобождении из-за ложного положительного результата — это катастрофа.

F-score — это средневзвешенное значение точности и отзывчивости, когда мы в равной степени обеспокоены обоими показателями.

```
Найдём данные значения для нашей тестовой выборки:
Test Accuracy score = 68.403%
Test Precision score = 69.174%
Test Recall score = 68.875%
Test F-score score = 68.6%
```

Довольно сбалансированные значения по всем направлениям - примерно одинаковый процент ложноположительных и ложноотрицательных результатов.

K-Fold кросс-валидация

Чтобы подчеркнуть важность правильной проверки модели и получить лучшее представление о производительности нашей модели, мы собираемся использовать K-Fold кросс-валидация. Мы разделяем наш обучающий набор данных на K уникальных наборов проверки - разделение обучения определяется K, где каждый набор проверки = (100 / K)% от всего набора данных, обучающий набор состоит из оставшихся наборов проверки K-1. Термин перекрестная проверка относится к проверке модели на нескольких наборах проверки.

Терминология здесь может быть несколько запутанной, потому что мы обычно разделяем наш 80%-ный обучающий набор на обучающий и тестовый набор на каждой итерации K- Fold CV. Мы оставляем за собой фактический набор тестов - тот, который мы создали изначально с разделением всего набора данных на 80/20 - для проверки производительности нашей модели после настройки ее гиперпараметров.

Таким образом, K-Fold CV будет обучать и оценивать K различных версий нашего классификатора. Обратите внимание, что в то время, как обучающие наборы перекрываются, наборы проверки никогда не перекрываются. Мы будем использовать 10-кратное резюме, при этом K = 10 является выбором, дающим средние оценки моделей с довольно низким уклоном и умеренной дисперсией, отчасти из-за результирующего соотношения обучения /проверки 90/10.

K-Fold CV обучает нашу модель на K = 10 различных, перекрывающихся тренировочных сгибах и проверяет ее производительность на K = 10 проверочных сгибах.

Поскольку мы обучаем классификатор, мы используем StratifiedKFold, который сохраняет процент выборок в каждом классе (эмоция) для каждого сгиба. Хотя у нас есть сбалансированный набор данных, стратифицированное K-кратное резюме особенно важно при классификации несбалансированного набора данных. Мы также устанавливаем shuffle=True для изменения порядка выборки классов в каждом сгибе в соответствии с поведением по умолчанию train_test_split от sklearn, чтобы мы могли точно сравнить это с K-кратным показателем CV.

```
KFold CV scores for MLP:
65.52%
62.93%
64.35%
72.17%
72.17%
55.65%
65.22%
72.17%
72.17%
66.96%
Среднее значение KFold CV for MLP: 66.93% ± 5.15%
```

Действительно, немного хуже, но это более точная оценка этой модели. Поскольку мы получаем значительно худшую производительность, подгоняя нашу модель к случайным подмножествам наших обучающих данных, мы могли бы предположить, что производительность нашей модели завышается из-за переобучения, когда мы тренируем и тестируем ее только один раз при обычном разделении 80/20.

Хотя K-кратное резюме требует больших вычислительных затрат, мы получаем гораздо больше информации из наших данных, и это серьезное преимущество,

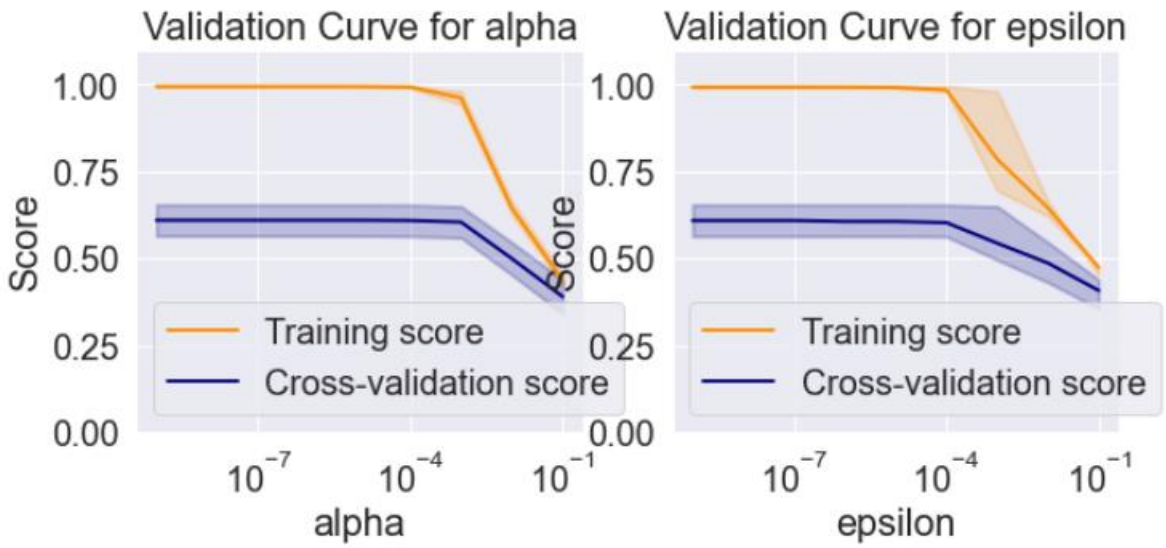
когда у нас очень мало обучающих выборок. Настройка модели только на один набор проверки, например, при разделении 60/20/20, может привести к искусственно завышенным показателям производительности, которые разочаруют, когда модель будет применена к реальным данным.

Кривая проверки

С помощью кривой проверки мы можем дополнительно проверить нашу модель на предмет переобучения, зависящего от гиперпараметров, и какое значение какого гиперпараметра может способствовать переобучению, указав диапазон параметров, по которым кривая проверки будет перекрестно проверять модель. Мы хотим выяснить, является ли наша модель переоснащенной (высокая дисперсия) или недостаточно подходящей (высокая предвзятость), или же мы достигли подходящего компромисса между дисперсией и предвзятостью. Опять же, наша модель переоснащена из-за ее отличной производительности при обучении и низкой производительности при перекрестной проверке. Увеличение параметра регуляризации alpha способствует уменьшению весов MLP и уменьшает способность нашей модели подгонять свои веса к данным обучения, поэтому мы посмотрим, сможем ли мы добиться прогресса в этом.

Кривая проверки - это своего рода поиск по сетке с одним параметром, и поэтому мы не ожидаем, что сможем лучше настроить наши гиперпараметры на основе того, что она нам говорит. Мы могли бы на самом деле сказать grid search, чтобы он возвращал как свое резюме, так и результаты обучения, и построить их оттуда, чтобы понять, как каждый гиперпараметр влияет на прогнозы нашей модели, но вычисление результатов обучения для сетки довольно дорого с точки зрения вычислений и на самом деле не обязательно.

Кривая проверки Sklearn реализует Stratified K-Fold CV для оценки модели для многоклассовых задач, поэтому мы зададим K = 10.



Как и ожидалось, здесь мало чему можно научиться, потому что поиск по сетке хорошо сработал для нас. Гиперпараметры просто хороши там, где они есть, и на самом деле не имеют большого значения в разумных пределах. Тем не менее, большой разрыв между оценкой обучения и оценкой CV является четкой визуализацией высокой дисперсии этой модели. Если бы кривая валидации была более показательной, мы бы искали, где кривые оценки обучения и CV находятся ближе всего.

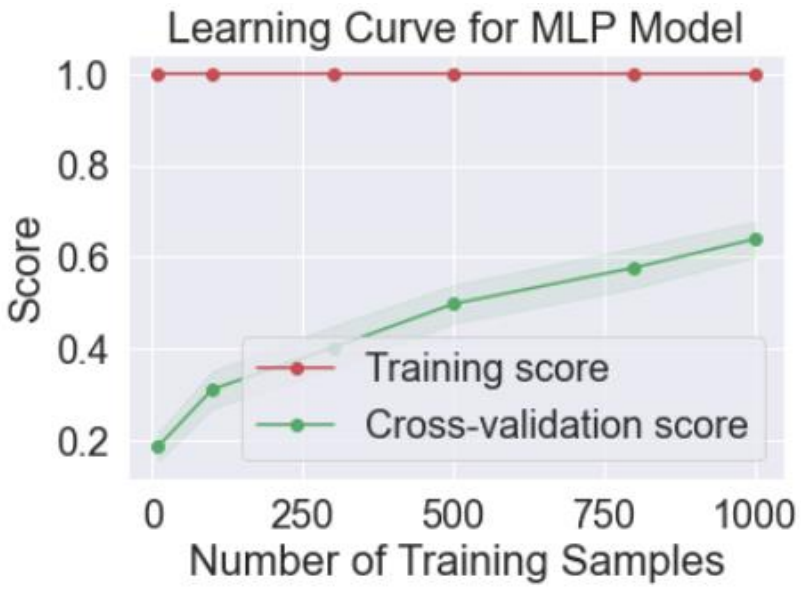
Поскольку методы глубокого обучения, как известно, лучше работают с большими наборами данных, давайте проверим, может ли размер нашего набора данных быть проблемой.

Кривая обучаемости

Мы можем использовать кривую обучения, чтобы сравнить производительность нашей модели, если бы мы обучали ее на меньших подмножествах нашего обучающего набора данных, и, таким образом, получить представление о том, можем ли мы ожидать увеличения производительности за счет использования большего количества обучающих данных или гораздо менее вероятно, что нам не нужно столько данных.

Мы указываем переменные размеры обучающих наборов, которые будут использоваться для кривой обучения, чтобы создать одну модель для каждого

размера. Помните, что, поскольку мы используем 0,8 * 1440 = 1152 выборки в нашем обучающем наборе, это верхняя граница, которую кривая обучения может проверить для нас. Как и кривая валидации, кривая обучения sklearn реализует Stratified K-Fold CV для оценки моделей многоклассовой классификации, поэтому мы снова указываем K = 10.



Одна вещь, о которой говорит нам кривая обучения, заключается в том, что размер набора данных не является нашей самой большой проблемой - наша точность находится на плаву и не принесет серьезной пользы от большего обучающего набора - возможно, максимум 5% при гигантском наборе данных. Опять же, разрыв между кривой оценки обучения и кривой оценки перекрестной проверки показывает нам, что модель имеет чрезвычайно высокую дисперсию и - она отлично работает с данными обучения, но плохо работает с перекрестной проверкой, потому что она серьезно перегружена - модель вообще плохо обобщает тестовые данные.

Стало ясно, что MLP может быть не лучшим выбором модели для этой задачи - похоже, она не обладает той сложностью, которая нам потребовалась бы для правильного различения различий между нашими функциями и эмоциями. На данный момент кажется, что нам потребуется значительно более сложная глубокая нейронная сеть, чтобы повысить производительность этого набора данных.

Заключение

В данной работе были рассмотрены различные модели машинного обучения, которые применялись для определения эмоционального состояния человека по записи его голоса. Проанализировали качество моделей, а также реализовали нейронную сеть - Многослойный персептрон (MLP). Изучили её и использовали различные инструменты для улучшения качества модели.

MLPClassifier является мощным в том смысле, что он обеспечивает заметную производительность при относительно небольших усилиях, затрачиваемых на исследовательский анализ, оптимизацию гиперпараметров и архитектуру модели; особенно с использованием преимуществ поиска по сетке, и тем более, когда мы знаем, как настроить каждый гиперпараметр индивидуально.

Нам придется изучить более сложные методы глубокого обучения, чтобы получить реальную производительность на этом наборе данных. Рекуррентные нейронные сети с долговременной кратковременной памятью (LSTM RNN) и Сверточные нейронные сети (CNN) являются отличными кандидатами DNN для классификации аудиоданных: LSTM RNN из-за их превосходной способности интерпретировать последовательные данные, такие как форма звукового сигнала, представленная в виде временного ряда; CNN, поскольку функции, разработанные на основе аудиоданных, такие как поскольку спектрограммы имеют заметное сходство с изображениями, в которых CNN преуспевают в распознавании и различении различных паттернов.

Список использованных источников

Андреас Мюллер, Введение в машинное обучение с помощью Python [Руководство для специалистов по работе с данными] / Андреас Мюллер, Сара Гвидо, – Москва, 2016-2017. – 393 с.

Коротеев М.В. Об основных задачах дескриптивного анализа как подготовительного этапа машинного обучения - 2018.

Коротеев М.В. Учебное пособие по дисциплине «Анализ данных и машинное обучение» - Москва, 2018.

Шакла Нишант, Машинное обучение и TenserFlow. – СПб.: Питер, 2019. – 336 с.: ил. – (Серия «Библиотека программиста»).

Воронцов К.В. Лекции по методу опорных векторов [Электронный ресурс].

URL: http://www.ccas.ru/voron/download/SVM.pdf (дата обращения 20.11.2020)

Приложения

Результирующие файлы:

Мерзляков ДА (191744) [ПИ19-3] Машинное обучение в задачах распознавания голоса.docx

Мерзляков ДА (191744) [ПИ19-3] Машинное обучение в задачах распознавания голоса.ipynb