# Esplorazione del suono e classificazione con Machine Learning

Sergio Cucinotta

4 settembre 2024

## 1 Introduzione

In questo progetto ho sviluppato un sistema per l'analisi dei suoni, che potrebbe essere esteso nell'ambito industriale per comprendere e individuare lo stato di usura e deterioramento nei macchinari.

L'obiettivo principale era di creare un modello in grado di rilevare le variazioni nei suoni prodotti dalle macchine e associarle a diverse condizioni, utilizzando suoni singoli di ritmica come proxy. Per raggiungere questo obiettivo, ho seguito un approccio che ha coinvolto diverse fasi, dalla preparazione dei dati all'implementazione e valutazione di un modello di machine learning.

In particolare, ho estratto caratteristiche rilevanti dai suoni utilizzando Librosa, ho applicato algoritmi di clustering per identificare pattern nei dati audio e infine ho addestrato un modello di classificazione, utilizzando il Support Vector Classifier (SVC, o Support Vector Machine in generale). L'efficacia del sistema è stata valutata attraverso l'analisi di metriche di valutazione e la visualizzazione dei risultati tramite matrici di confusione. Gli ottimi risultati ottenuti con SVC non hanno reso necessario l'utilizzo di modelli complessi, come le CNN, per questo progetto incentrato su one-shots.

Nel documento presenterò nel dettaglio il processo seguito e i risultati ottenuti, con l'obiettivo di fornire un'analisi esaustiva delle potenzialità di questo approccio.

# 2 Preparazione dei Dati

## 2.1 Estrazione dei dati audio

Ho creato una funzione denominata extract_audio_paths(dir_path) per recuperare i percorsi dei file audio presenti nella directory specificata. Vengono accettati solo file .wav e vengono ignorati quelli che iniziano con "._" e altri file che non vengono letti correttamente.

```
def extract_audio_paths(dir_path):
    audio_paths = []
    for root, dirs, files in os.walk(dir_path):
        for file in files:
            if file.endswith(".wav") and not file.startswith("._")
                :
                audio_path = os.path.join(root, file)
                try:
                    librosa.load(audio_path, sr=None)
                    audio_paths.append(audio_path)
                except (Exception, librosa.LibrosaError) as e:
                    print(f"Warning: Error loading file '{
                        audio_path}': {e}")
    return audio_paths

audio_dir = Path("D:\SoundExploration")
audio_paths = extract_audio_paths(audio_dir)
```

## 2.2 Estrazione delle caratteristiche audio

Per estrarre le caratteristiche audio rilevanti da ciascun file ho utilizzato la libreria Librosa nella funzione extract_features. Le caratteristiche estratte includono i coefficienti MFCC, Spectral Centroid e Zero-crossing rate: le variazioni nei coefficienti MFCC riflettono le distorsioni spettrali, mentre Spectral Centroid e ZCR forniscono indicazioni sulla luminosità e sulla presenza di transizioni rapide nel suono, che possono essere indicative di distorsioni nel segnale audio.

Queste caratteristiche offrono quindi una rappresentazione significativa della distorsione del suono, consentendo un'analisi dettagliata delle variazioni acustiche.

```
def extract_features(audio_path):
    y, sr = librosa.load(audio_path)
    n_fft = min(2048, len(y))
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=6, n_fft=n_fft
        , n_mels=128)
    spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=
        sr, n_fft=n_fft)[0]
    zcr = librosa.feature.zero_crossing_rate(y)[0]
```

```
    return np.concatenate([mfccs.mean(axis=1), [np.mean(
        spectral_centroid)], [np.mean(zcr)]])

features = []
for audio_path in audio_paths:
    features.append(extract_features(audio_path))
```

### 2.3 Scalatura delle caratteristiche e riduzione delle dimensionalità

Dopo l'estrazione delle caratteristiche, ho applicato la scalatura Min-Max utilizzando il MinMaxScaler() per garantire che tutte le caratteristiche abbiano la stessa scala e, utilizzando la Principal Components Analysis (PCA), ho ridotto la dimensionalità delle caratteristiche estratte a due componenti principali. Questo mi ha semplificato il processo di clustering e migliorato le prestazioni del modello.

```
scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)

pca = PCA(n_components=2)
reduced_features = pca.fit_transform(features_scaled)
```

### 2.4 Clustering dei dati

Ho eseguito il clustering dei dati ridotti a due dimensioni utilizzando l'algoritmo K-Means con tre cluster distinti. L'obiettivo è stato quello di raggruppare i dati in base alla loro somiglianza, identificando pattern e strutture nel dataset audio.

Ho selezionato 3 cluster per una suddivisione chiara dei dati. Limitando le iterazioni a 300, si assicura una convergenza entro un numero ragionevole di passaggi. Con 10 inizializzazioni diverse, si aumenta la probabilità di trovare il miglior set di centroidi. Il settaggio di 'random_state=0' assicura la replicabilità dei risultati.

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300,
    n_init=10, random_state=0)
kmeans.fit(reduced_features)
```

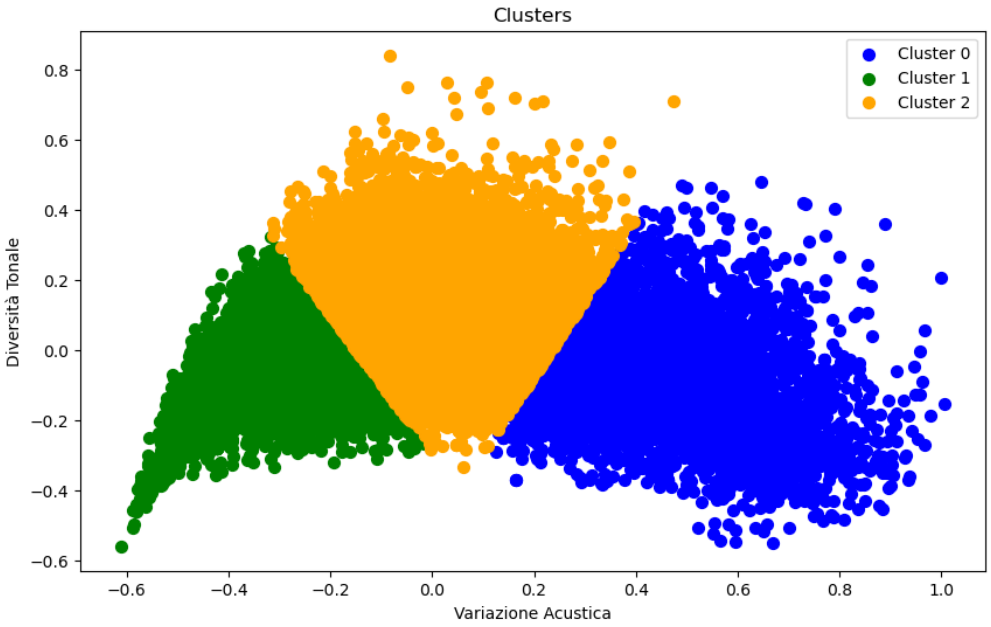
### 2.5 Raggruppamento e Visualizzazione dei cluster

Successivamente ho creato un dizionario denominato clusters per raggruppare i percorsi dei file audio in base alle etichette dei cluster ottenute dall'algoritmo KMeans. Utilizzando un ciclo for, ogni percorso dei file audio viene associato alla sua etichetta di cluster corrispondente nel dizionario clusters. Così ho potuto visualizzare i cluster e la distribuzione delle varie tipologie di one-shots all'interno dei cluster.

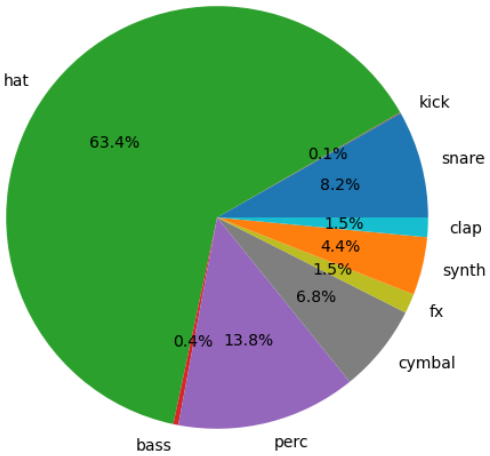
```python
clusters = {}
for i, label in enumerate(kmeans.labels_):
    if label not in clusters:
        clusters[label] = []
    clusters[label].append(audio_paths[i])

for cluster_label, file_paths in clusters.items():
    count = 0
    for file_path in file_paths:
        count += 1
    print(f"Cluster {cluster_label}: {count} samples")
```

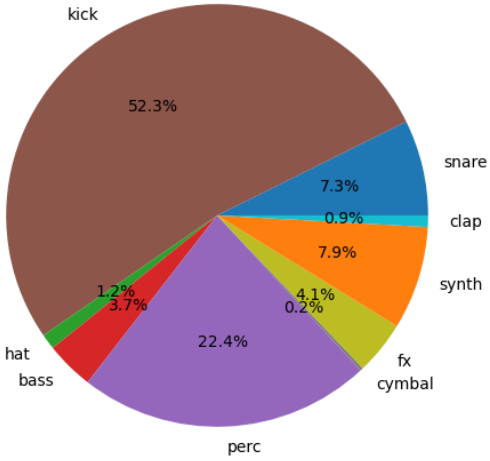
Cluster 0: 5714 samples
Cluster 1: 8458 samples
Cluster 2: 11417 samples

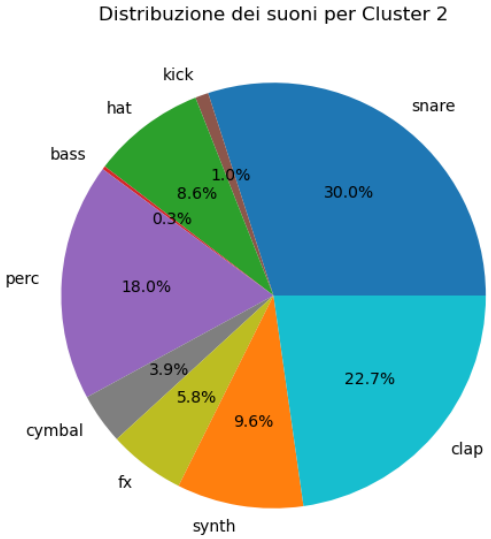


Distribuzione dei suoni per Cluster 0




Distribuzione dei suoni per Cluster 1






Nell'analisi dei cluster ottenuti è emersa una suddivisione significativa basata sulla variazione dell'intensità nel tempo. I tre cluster identificati evidenziano differenti caratteristiche sonore e suggeriscono la presenza di pattern distinti all'interno del dataset audio:

- Cluster 0 (Variazione Acustica $> 0.2$):

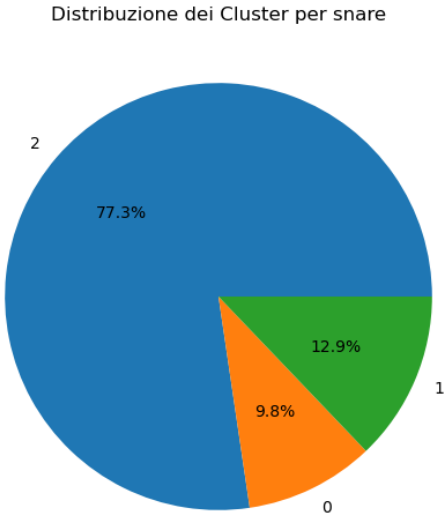
  In questo cluster ho notato una variazione acustica positiva con valori superiori a 0.2. La presenza predominante di suoni come gli hi-hat suggerisce infatti che questo cluster potrebbe contenere sample con intensità elevate e un'ampia variazione dinamica, tipica di suoni percussivi ad alto impatto.

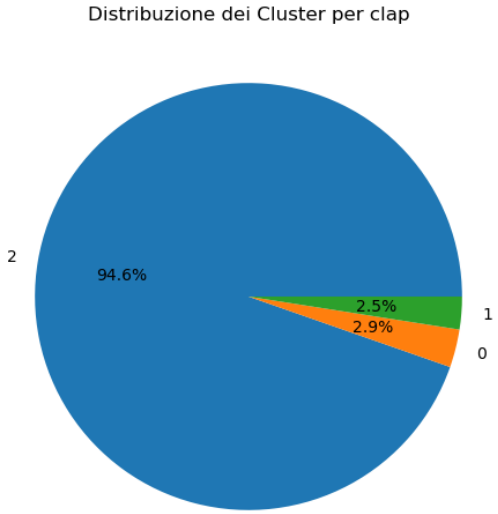
- Cluster 1 (Variazione Acustica Negativa):

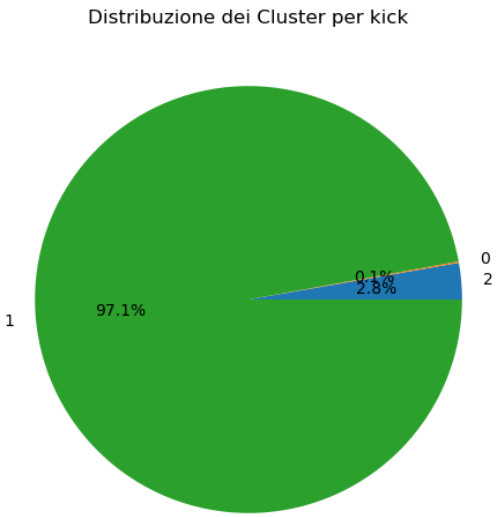
  Nel cluster 1, ho osservato una variazione acustica negativa, indicando una diminuzione dell'intensità nel corso del tempo. La presenza predominante all'interno di questo cluster è infatti quella dei kick,sample con un forte attacco seguito da un rapido decadimento, caratteristico dei suoni percussivi bassi e potenti.

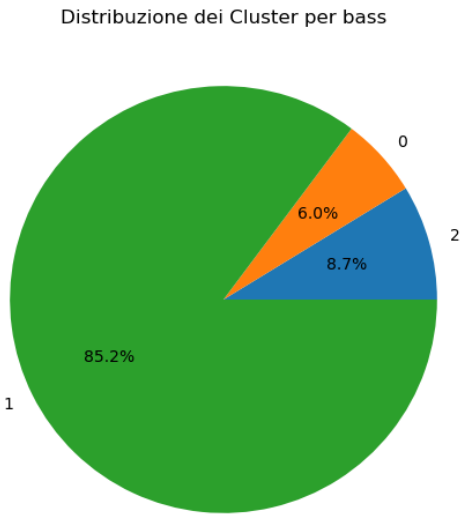
- Cluster 2 (Variazione Acustica Intorno allo 0):

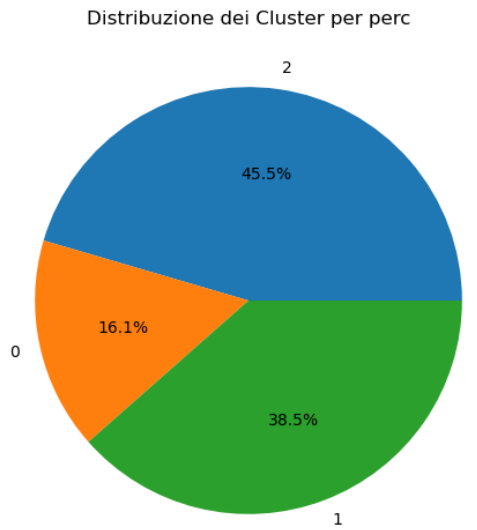
  Nel cluster 2, ho riscontrato una variazione acustica intorno allo zero. Questo suggerisce che i sample in questo cluster potrebbero mostrare una distribuzione uniforme di intensità nel segnale audio. È interessante notare che questo cluster contiene principalmente sample dello snare, che tendono ad avere una presenza sonora moderata e un'ampia varietà di frequenze.

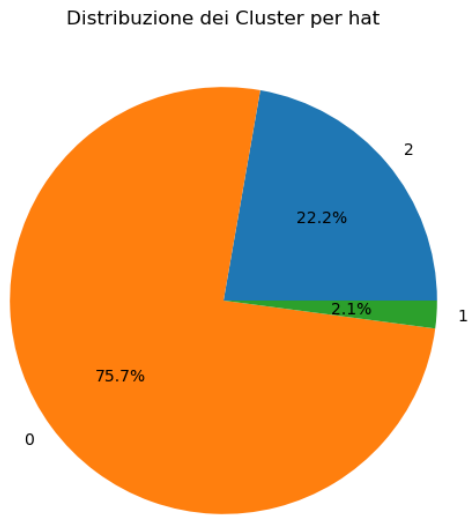

Distribuzione dei Cluster per snare
2
77.3%
12.9%
1
9.8%
0

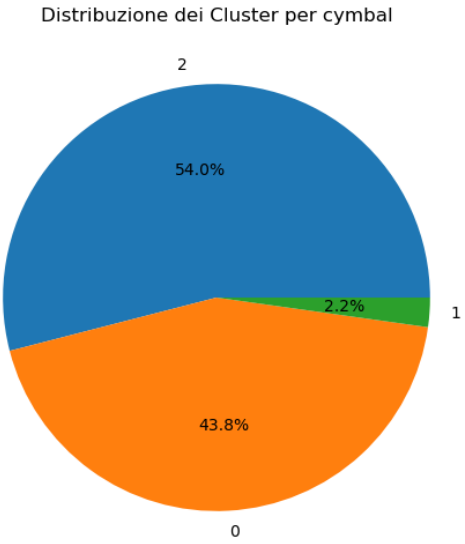

Distribuzione dei Cluster per clap
2
94.6%
2.5%
1
2.9%
0

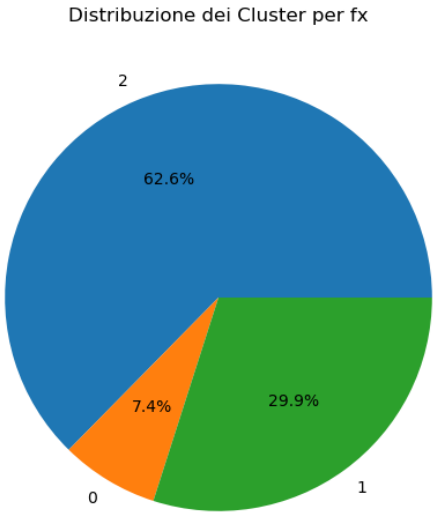

Distribuzione dei Cluster per kick
0
0.1%
2
2.8%
1
97.1%

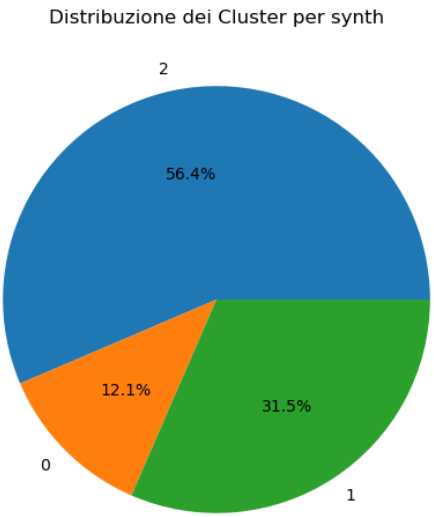

Distribuzione dei Cluster per bass
0
6.0%
2
8.7%
85.2%
1


Distribuzione dei Cluster per perc
2
45.5%
16.1%
0
38.5%
1


Distribuzione dei Cluster per hat
2
22.2%
2.1%
1
75.7%
0


Distribuzione dei Cluster per cymbal
2
54.0%
2.2%
1
43.8%
0


Distribuzione dei Cluster per fx
2
62.6%
7.4%
29.9%
0
1


Distribuzione dei Cluster per synth
2
56.4%
12.1%
31.5%
0
1

# 3 Sviluppo del Modello

## 3.1 Scelta e Training del Modello

Per lo sviluppo del modello ho scelto l'SVC, utile per l'adattabilità ai dati clusterizzati. Ho diviso i dati in un set di training e un set di test utilizzando la funzione train_test_split. Ho quindi addestrato il modello utilizzando il metodo fit con i dati di addestramento X_train e le relative etichette y_train. In pratica, durante il processo di addestramento, il modello impara a relazionare le features fornite alle rispettive etichette target.

Ho utilizzato la validazione incrociata per stimare le prestazioni del modello su dati non visti e ho calcolato i punteggi di validazione incrociata utilizzando la funzione cross_val_score, valutando così il modello su più suddivisioni dei dati e calcolando la media dei punteggi.

I risultati ottenuti dall'SVC hanno reso superfluo l'utilizzo di modelli più complessi come le reti neurali.

```
model = SVC()

X_train, X_test, y_train, y_test = train_test_split(
    reduced_features, kmeans.labels_, test_size=0.2, random_state
    =42)

model.fit(X_train, y_train)

cv_scores = cross_val_score(model, reduced_features, kmeans.
    labels_, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", np.mean(cv_scores))
```

## 3.2 Metriche di valutazione e visualizzazione

Dopo l'addestramento del modello, ho calcolato diverse metriche per valutare le sue prestazioni sui dati di test. L'accuratezza, la precisione, il recall e il punteggio F1 sono stati calcolati utilizzando le funzioni fornite da scikit-learn.

```
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Cross-Validation Scores:
[0.9982415 0.99882767 0.99785072 0.99882767 0.99941372]
Mean Cross-Validation Score: 0.9986322553231333
Accuracy: 0.9986322782336851
Precision: 0.9986331558587084
Recall: 0.9986322782336851
F1 Score: 0.9986323669727319

Le metriche rilevanti, i punteggi di validazione incrociata e la confusion matrix indicano che il modello SVC ha ottenuto prestazioni eccezionali nella classificazione dei dati: i punteggi di validazione incrociata mostrano che il modello è generalmente robusto e coerente su diverse partizioni dei dati e le alte misure di accuratezza, precisione, recall e punteggio F1 suggeriscono che il modello è in grado di fare previsioni precise e affidabili sui 3 cluster

In conclusione, i risultati complessivi evidenziano l'efficacia e l'affidabilità del modello SVC nella classificazione dei dati ritmici, come confermato anche dalla confusion matrix.