AI capstone project 1 report

111550060 劉千慈

GitHub link: https://github.com/liucht4212/AI_project1.git

1. Research Questions

在學習日文的過程中，台灣人常會有難以分辨濁音及清音的時刻，如：た、だ和か、が等，主要的原因是這些音在發音時，主要的區別在於聲帶振動與否，而非送氣強度（如：英文的 p 跟 b 等），然而，不同的說話者所發音及錄音設備不同，分辨難度會更為上升。

此研究主要探討如何利用機器學習，從日語語音訊號中準確區分濁音與清音。並且做下列實驗：

a. 比較不同音訊特徵提取（如：mfcc、zero crossing rate 等）
b. 比較不同機器學習模型（如：SVM、Random Forest 等）
c. 比較不同數據數量及處理（如：數據平衡、訓練集數量等）

2. Dataset Document:

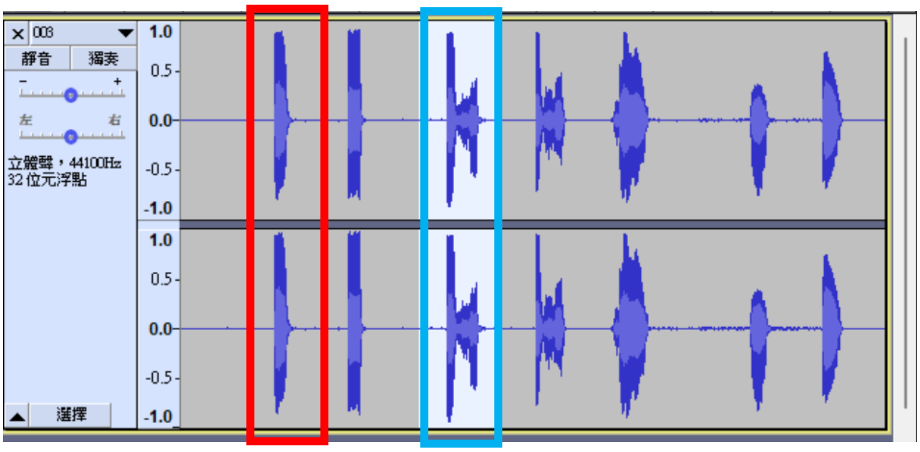
a. 資料類別：皆為 mp3 檔案，以 voiced（濁音）、voiceless（清音）兩個資料夾分別儲存。
b. 資料來源：致良出版社的《新 e 世代日本語 1》教材音檔（已取得授權）以及自己與旅日朋友的語音資料。
c. 資料組成：內容組成有單音、單詞及少許句子，分類分布如下：

	音檔清音	自錄清音	音檔濁音	自錄濁音
數量	175	20	119	15
總體占比	53.2%	6.1%	36.2%	4.5%
類別占比	59.3%		40.7%	

註：自己錄音都會在單獨類別中佔有約 10%

d. 資料蒐集及處理：
   (1). 蒐集及處理：用手機、電腦錄音，手機錄音檔案為 m4a 檔，使用 FFMPEG 轉檔為 mp3 檔案後，使用 Audacity 剪輯
   (2). 分類標準：日語中還有半濁音，這部分不納入資料，半濁音會有送氣強度的分別，非主題探討的部分。
      i. 單音：依照日語定義直接分類為清音或濁音。
      ii. 單詞或句子：若包含一個濁音，則歸類為濁音。

e. 範例：



以上圖為例，經由聽且對照教材，紅框者為あ，則剪集此段片段（通常為 1~1.5 秒）儲存為 あ.mp3 歸類在 voiceless 中，藍框者為單字（あし），亦是歸類在 voice 中；另一舉例，是そで，因為有で，則會剪輯並存至 voiced，而資料集內無任何重複資料。

3. Method and expected result:

I. Feature Extraction:

i. 方法：

以語言學說法，分辨清濁音的方法並非照羅馬拼音中的て跟で念作送氣的 te 跟不送氣的 de，而是得靠喉嚨震動去辨別，又或者照秋山燿平在其 youtube 影片中歸納出的通則，如果在單詞開頭則有送氣與不送氣之分，非開頭者則濁音可能有鼻音之變化（秋山燿平，2022），因此在特徵提取，要選取貼近人耳辨識以及能辨識喉嚨震動者。

在此專題引用了 librosa 中，而只能讀取 wav 檔，所有音訊均標準化為 16kHz 單聲道 wav 格式，以確保一致性。

a. Mel Spectrogram：

主要是將音檔分格作傅立葉轉換，獲得頻率上的特徵，再藉由其他轉換得到能量分布，通常維度較高，在此為 40 維，且取每音檔分格平均作為單一音檔特徵。

b. MFCC：

和 Mel Spectrogram 是類似做法，只是又有作對數並作 DCT 能得到維度較低的資料，也是現今最常用的方式，在此研究有 13 維，亦是取每音檔分格平均作為單一音檔特徵。

c. Zero-crossing rate：

主要是計算音訊訊號穿過零軸的頻率，通常反映了聲音的「粗糙度」，而清濁音聲帶振動的不同可能會造成穿過零軸的頻率不同，也是常見的特徵提取方法，維度只有 1 維。

ii. 預期結果：

根據各特徵的維度來看，MFCC 的維度可能是最適合，因為 Mel Spectrogram 維度太高容易 overfitting，而 Zero-crossing rate 特徵數太少，可能會不足以達到好的預測。

因此預測表現效果：MFCC > Mel Spectrogram > Zero-crossing rate。

II. Supervised Learning:

使用了 sklearn 內的 SVC 和 RandomForestClassifier 做為訓練模型，並用 5-fold Stratified 交叉驗證，使用 Accuracy、AUROC、F1 score、Precision、Recall 作為評斷。

a. SVM (Support vector machine)

使用線性 SVM 找到清濁音的分類邊界，主要是猜測清音濁音二元問題能在高維度特徵找到線性分界。

- Hyperparameter 設定：
  (1). Kernel: Linear，因為是二元分類問題，且特徵維度較高，使用線性效率較快並且簡單。
  (2). Class weight：不設定，由於資料量較少，且整體分類比例約為 6:4，因此不考慮資料平衡處理。

b. Random Forest

因為 Random Forest 可適應非線性決策邊界，想以此作與線性 SVM 的比較。

- Hyperparameter 設定：
  (1). n_estimators 和 Max_depth：由於資料量少，因此設為較低的 45 棵樹及深度 5，以免 over-fitting。

- 預測結果：

根據資料組成及處理來看，音檔以固定的 50 音作為組成，沒有太多雜音，資料量少的情況下，使用線性的 SVM 應該可以得到更好的表現，Random Forest 可能難以透過選擇排除無關特徵。

III. Unsupervised learning

a. K-Means

使用 K-Means clustering 進行 Unsupervised learning，以評估提取的特徵是否能夠自然地將清濁音分類。由於類別只有 2 類，因此為 K = 2，並且因特徵維度有 40 及 18 者，以 PCA 降維以視覺化分群效果。

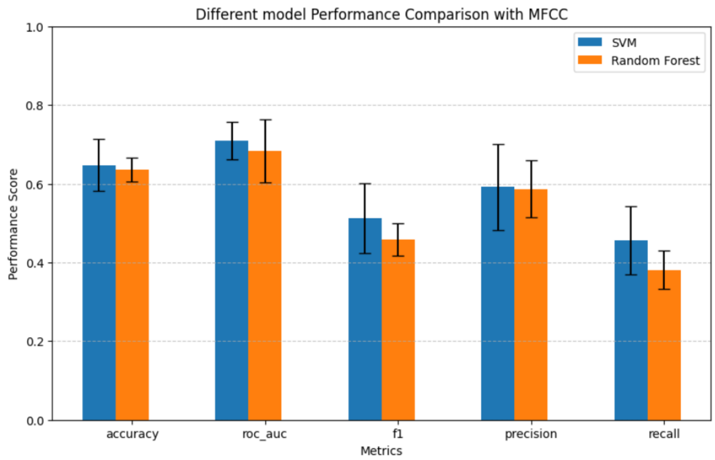
■ 預期結果：

由於清濁音的頻譜特徵應該具有一定的連續性，而 K-Means 主要基於距離度量，可能無法捕捉這些細微的模式，K-Means 因此難以很好地區分兩類。

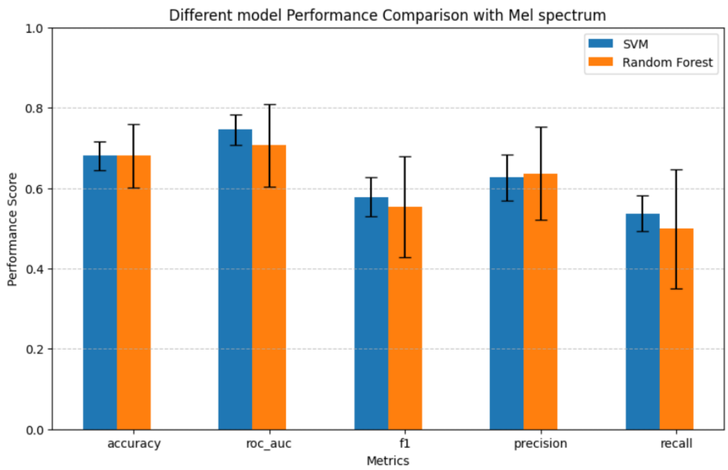
4. Experiment and analysis:

I. 不同模型表現

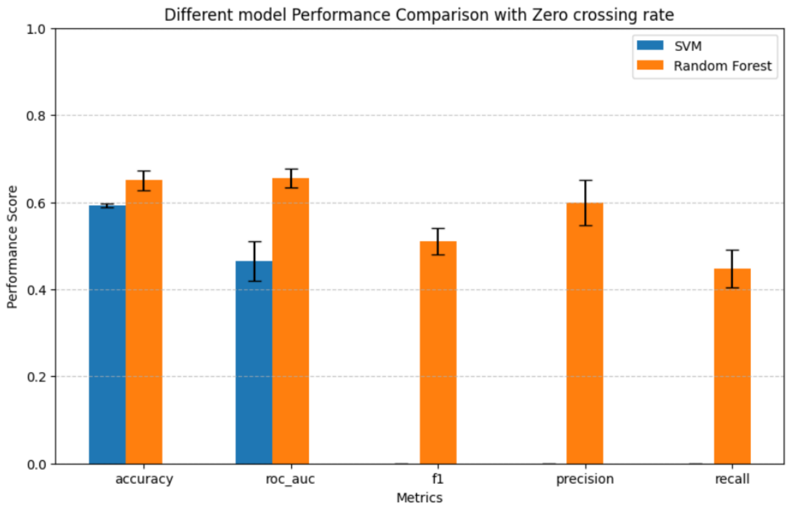
a. MFCC 特徵



b. Mel Spectrogram 特徵



c. Zero-crossing rate 特徵



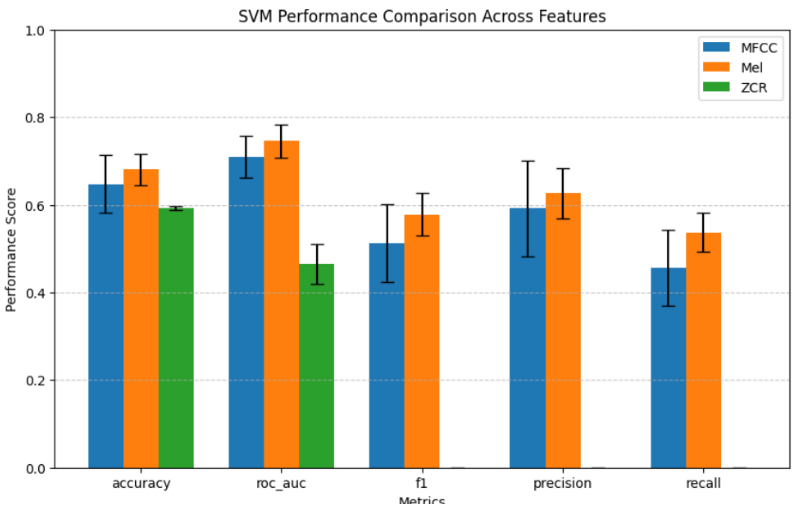
藉由上方結果，可以發現 MFCC 跟 Mel Spectrogram 兩者都是 SVM 表現優於 Random Forest。

其中，在 MFCC 較不明顯，但在 Mel Spectrogram 的 Random Forest 的標準差較大，可以驗證推測中在高維度且資料量少的情況下，Random Forest 較難有效形成結構較好的樹，因此較不穩定；而 SVM 較能適應這類型資料集，因此表現較好。

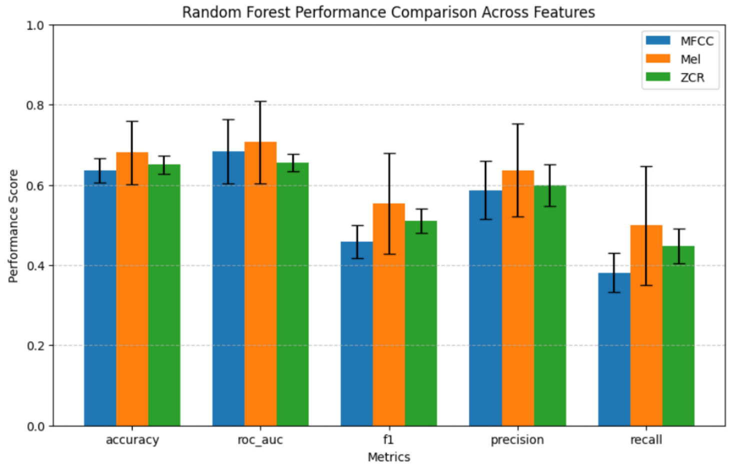
然而，在 Zero-crossing，卻是 Random Forest 優於 SVM，並且發生 F1 score、precision 跟 recall 為 0 的狀況，發現為 precision 除以 0，表示模型整體往同一方向（清音）猜，這可能是因為訓練集樣本不平衡導致，因此，將會多做處理不平衡狀況。

II. 不同特徵提取對於此任務的表現

a. SVM



b. Random Forest



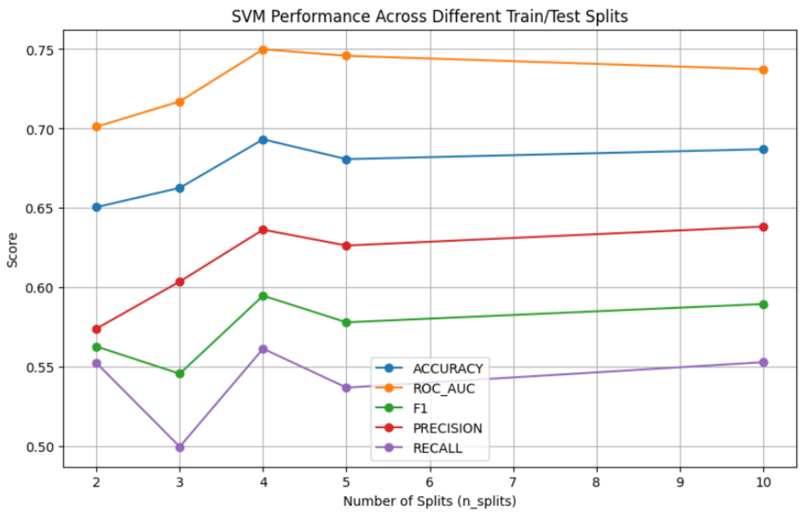
藉由圖表可以發現 Mel Spectrogram 都是表現最好的，在 SVM 中，第二好者為 MFCC；然而在 Random Forest 中反而是 zero-crossing rate 為第二好者，雖然差距並不大。這與原本的推測都不同。

可能是因為 Mel Spectrogram 保留較完整的特徵，能讓音頻中清濁音的分別能更好的保留下來，使得模型能更好學到；而 zero-crossing rate 較好可能是，因為維度較小，在 Random Forest 較能找到穩定結構的樹。

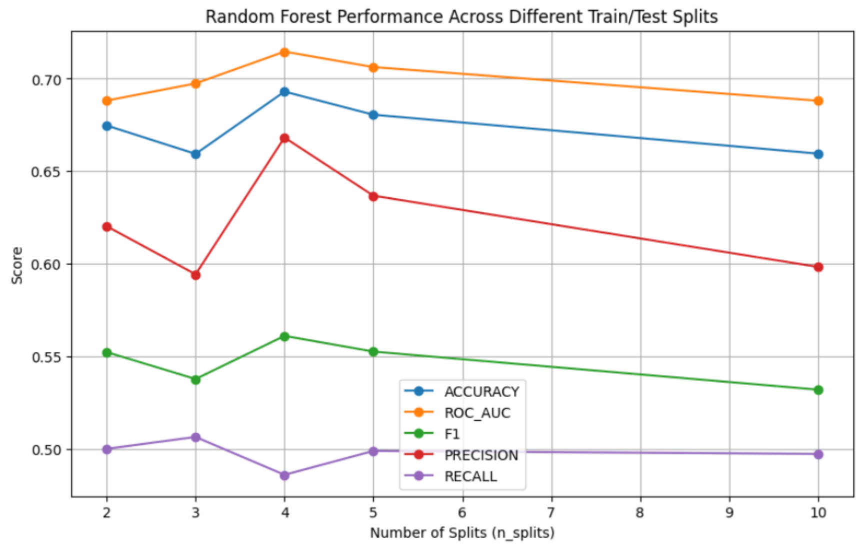
III. 訓練集大小對於此任務的表現

在此會以不同分等份數去使用 cross-validation 去作實驗，而等份數有 2, 3, 4, 5, 10，由於 2、3 可能學習不夠，然而在 10 可能又會難以泛化，因此，推測會在 4~5 者達到高峰，以先升後降的趨勢呈現。

a. SVM



b. Random Forest

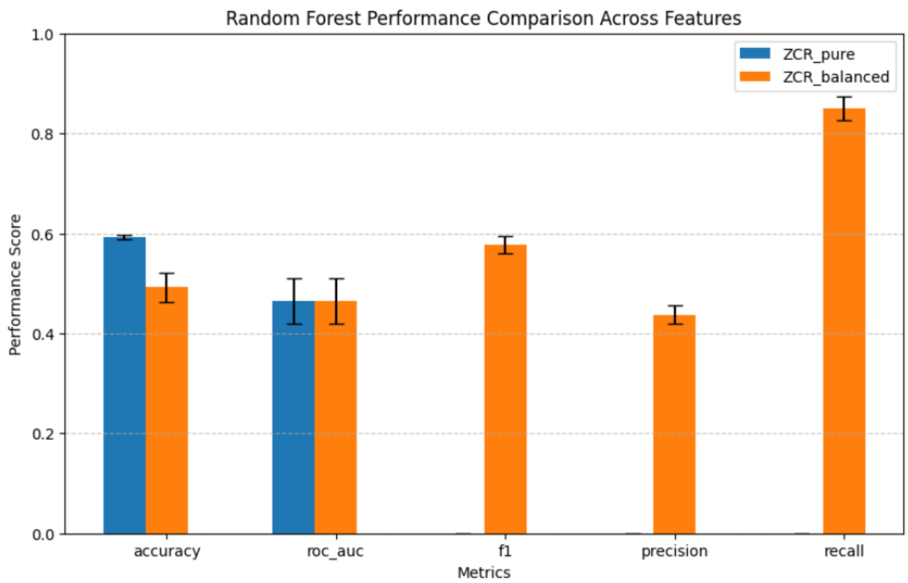


大致上符合推測，然而在兩者的等份數為 3 者卻有稍微下降趨勢，我認為可能是因為此 Random seed 在切分時會切出較難猜測的測試集，應該要以更多不同的 Random seed 做更多實驗求其平均與標準差，才能得到更客觀的結果。

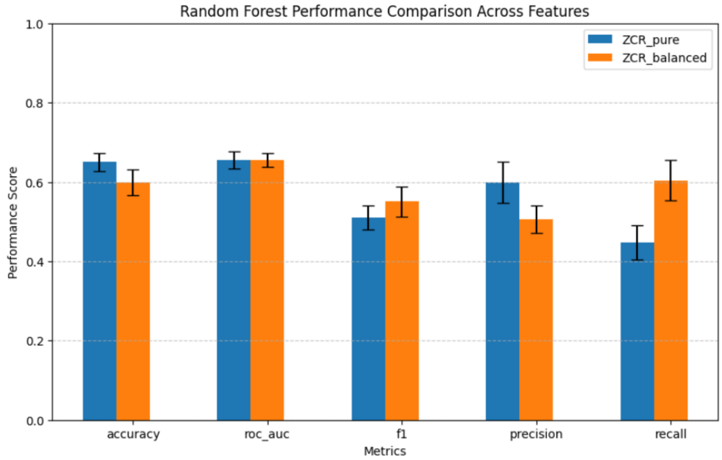
IV. 數據平衡

由於資料比例約為 6：4，在此沒有使用資料生成（如：SMOTE），又或者是 under sampling 等幫助平衡資料，而是在訓練過程中加入 class_weight 幫助不平衡資料的權重提高。

a. SVM



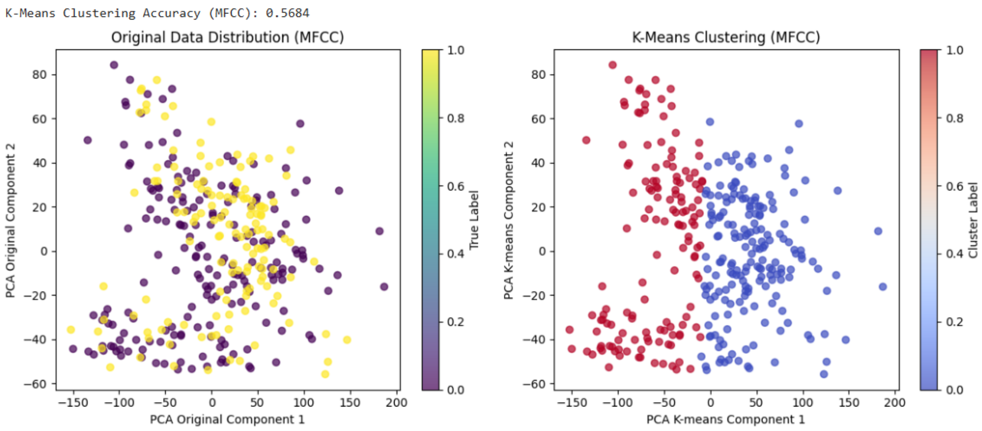
b. Random Forest



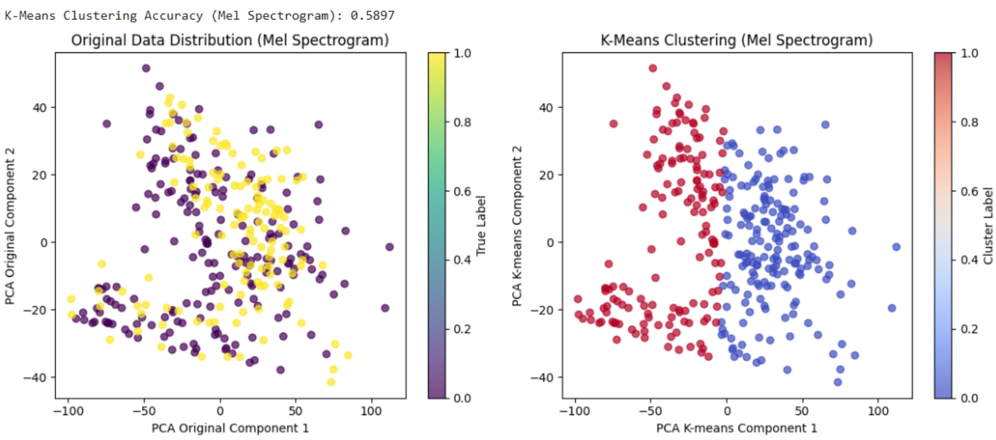
訓練時偏重減少於較少資料者的 Loss，即會讓另一邊準確率較低，因此整體正確率會下降（因無腦猜較多者會有接近 60%的正確率），但 F1 score、precision 跟 recall 會上升。

V. Clustering 結果：

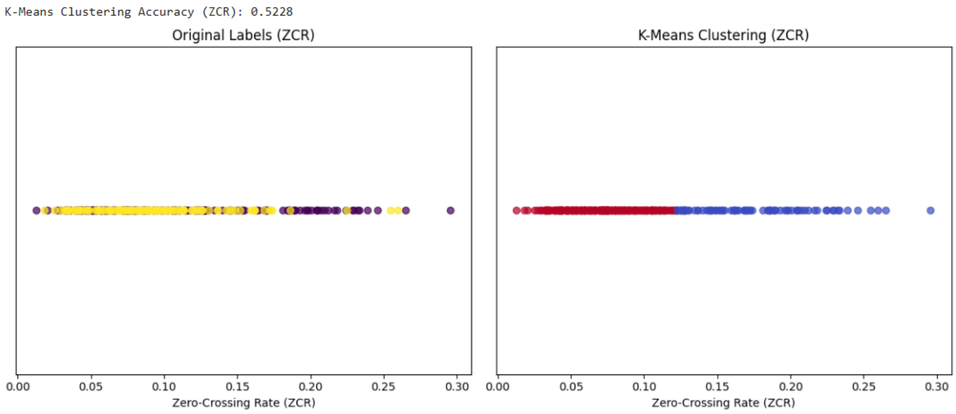
a. MFCC



b. Mel Spectrogram



c. Zero-crossing rate



準確率分別為：MFCC 為 0.5684，Mel Spectrogram 為 0.5897，ZCR 為 0.5228。表現結果不太好，可以驗證推測中，認為音訊特徵為連續且細微者，難以利用 K-means 此用資料點與點的距離的方法去發現音的變化，可能發現更多的是單詞組成的不同。

5. Discussion:

實驗結果與預期結果相比，大多符合預測，SVM 的確在高維度特徵且資料量少的狀況比 Random Forest，但未考慮到的是 Random Forest 在低維度特徵（Zero-crossing rate）能比 SVM 更適配；在訓練集大小跟不平衡實驗也大致符合預測。然而，在特徵提取的部分，會認為維度適中者最好，但最後的結果卻是維度最大者最好，表示在這個議題中仍得保持更完整的特徵會更好。

而經過實驗後可以發現影響表現的因素有：

a. 特徵提取：保留越完整的特徵越好，由於聲音檔案的變化較細微。
b. 模型選擇：根據特徵提取的特性去選擇適合模型，如：高維特徵適合 SVM，低維特徵則更適合 Random Forest。
c. 資料平衡處理：因為語音資料組成較為複雜，即使整體分類趨近於平衡，但仍有可能讓模型難以學習類別較小者。

如果有更多時間的話，首先，我想先將訓練集的實驗更加完善，藉由改變資料切分的 Random Seed（至少取 30 個做平均與標準差）去更客觀知道完整的趨勢。然後，我覺得因為高維度特徵可能更適用於 Neural Network 模型，因此若能在特徵提取時將頻率特徵換為圖片去訓練 CNN 似乎也可行。又或者是利用一些數據增強方式（如：選取特定頻率段、拆解語音）去了解是哪種聲音的組成對於此問題更加重要。

在這個實驗中，我學到使用模型適用特性，以及特徵提取選擇的重要性，還有關於數據特性（如：數據大小、組成複雜性、類別比例）對結果的影響。但仍有一些問題存在，是否真的增加數據量 Random Forest 會更加穩定。又或者是更根基的問題，到底是哪一種特徵對於分辨清濁音是更加適合的。

6. Reference:

楊永良、林秀禧（2019）：《新 e 世代日本語 1》教材音檔。致良出版社。

秋山燿平（2022 年 9 月 23 日）：〈為什麼日文中的「か」聽起來像「ga」？日本人讓你徹底弄明白！〉〔影片〕。YouTube。https://youtu.be/WOscODPNytM?si=b2N2kh5NAljSkUSu

Scikit-learn（Pedregosa et al., 2010）: SVM 及隨機森林模型。

Librosa（McFee et al., 2013）: 音訊特徵 MFCC、Mel spectrogram 及 zero-crossing rate 提取。

7. Appendix:

Requirements

In [14]:
```
import numpy as np
import pandas as pd
import os
import librosa
import matplotlib.pyplot as plt
from pydub import AudioSegment
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate, StratifiedKFold
from sklearn.metrics import accuracy_score, roc_auc_score, average_precision_score, f1_score, precision_score, rec
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

Change the format of the input

In [ ]:
```
!git clone --depth 1 --filter=blob:none --sparse https://github.com/liucht4212/AI_project1.git
!cd AI_project1 && git sparse-checkout set voiced voiceless
```

In [ ]:
```
input_folder = "AI_project1"
output_folder = "processed_dataset"

for category in ["voiced", "voiceless"]:
    os.makedirs(f"{output_folder}/{category}", exist_ok=True)
    for file in os.listdir(f"{input_folder}/{category}"):
        if file.endswith(".mp3"):
            audio = AudioSegment.from_mp3(f"{input_folder}/{category}/{file}")
            audio = audio.set_frame_rate(16000).set_channels(1)
            audio.export(f"{output_folder}/{category}/{file.replace('.mp3', '.wav')}", format="wav")
```

Feature Extraction

In [4]:
```
dataset_folder = "processed_dataset"
np.random.seed(42)
X_mfcc = []
y_mfcc = []
X_zcr = []
y_zcr = []
X_mel = []
y_mel = []
X_contrast = []
y_contrast = []

for label, category in enumerate(["voiceless", "voiced"]):
    for file in os.listdir(f"{dataset_folder}/{category}"):
        if file.endswith(".wav"):
            y, sr = librosa.load(f"{dataset_folder}/{category}/{file}", sr=16000)
            mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
            mfccs_mean = np.mean(mfccs, axis=1)
            X_mfcc.append(mfccs_mean)
            y_mfcc.append(label)

            zcr = librosa.feature.zero_crossing_rate(y=y)
            zcr_mean = np.mean(zcr, axis=1)
            X_zcr.append(zcr_mean)
            y_zcr.append(label)

            mel_spec = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=40)
            mel_spec_db = librosa.power_to_db(mel_spec)
            mel_mean = np.mean(mel_spec_db, axis=1)
            X_mel.append(mel_mean)
            y_mel.append(label)


X_mfcc = np.array(X_mfcc)
y_mfcc = np.array(y_mfcc)
X_zcr = np.array(X_zcr)
y_zcr = np.array(y_zcr)
X_mel = np.array(X_mel)
y_mel = np.array(y_mel)
```

Supervised learning-MFCC

In [ ]:
```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
svm_model = SVC(kernel="linear", probability=True, random_state=42)
metrics = {"accuracy": [], "roc_auc": [], "f1": [],"precision": [], "recall": []}

for train_idx, test_idx in cv.split(X_mfcc, y_mfcc):
    X_train, X_test = X_mfcc[train_idx], X_mfcc[test_idx]
    y_train, y_test = y_mfcc[train_idx], y_mfcc[test_idx]
    svm_model.fit(X_train, y_train)
    y_pred = svm_model.predict(X_test)
    y_proba = svm_model.predict_proba(X_test)[:, 1]
    metrics["accuracy"].append(accuracy_score(y_test, y_pred))
    metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
    metrics["f1"].append(f1_score(y_test, y_pred))
    metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
    metrics["recall"].append(recall_score(y_test, y_pred))

svm_performance_mfcc = {metric: {"mean": np.mean(values), "std": np.std(values)} for metric, values in metrics.ite
print("===== SVM Cross-Validation Performance =====")
for metric, values in svm_performance_mfcc.items():
    print(f"  {metric.upper()}: {values['mean']:.6f} ± {values['std']:.6f}")
```

In [ ]:
```
# Random Forest Model
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
rf_model = RandomForestClassifier(n_estimators=45, max_depth=5, random_state=42)
metrics = {"accuracy": [], "roc_auc": [], "f1": [],"precision": [], "recall": []}

for train_idx, test_idx in cv.split(X_mfcc, y_mfcc):
    X_train, X_test = X_mfcc[train_idx], X_mfcc[test_idx]
    y_train, y_test = y_mfcc[train_idx], y_mfcc[test_idx]
    rf_model.fit(X_train, y_train)
    y_pred = rf_model.predict(X_test)
    y_proba = rf_model.predict_proba(X_test)[:, 1]
    metrics["accuracy"].append(accuracy_score(y_test, y_pred))
    metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
    metrics["f1"].append(f1_score(y_test, y_pred))
    metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
    metrics["recall"].append(recall_score(y_test, y_pred))

rf_performance_mfcc = {metric: {"mean": np.mean(values), "std": np.std(values)} for metric, values in metrics.item
print("===== Random Forest Performance =====")
for metric, values in rf_performance_mfcc.items():
    print(f"  {metric.upper()}: {values['mean']:.6f} ± {values['std']:.6f}")
```

In [ ]:
```
#comparison with two model
metrics = ["accuracy", "roc_auc", "f1", "precision", "recall"]
features = ["SVM", "Random Forest"]

means = np.array([
    [svm_performance_mfcc[m]["mean"] for m in metrics],
    [rf_performance_mfcc[m]["mean"] for m in metrics],
])
stds = np.array([
    [svm_performance_mfcc[m]["std"] for m in metrics],
    [rf_performance_mfcc[m]["std"] for m in metrics],
])

x = np.arange(len(metrics))
width = 0.25
fig, ax = plt.subplots(figsize=(10, 6))
for i in range(len(features)):
    ax.bar(x + i * width, means[i], width, label=features[i], yerr=stds[i], capsize=5)

ax.set_xlabel("Metrics")
ax.set_ylabel("Performance Score")
ax.set_title("Different model Performance Comparison with MFCC")
ax.set_xticks(x + width)
ax.set_xticklabels(metrics)
ax.legend()
plt.ylim(0.0, 1.0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

Supervised learning-Mel spectrogram

```
In [ ]: cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
        svm_model = SVC(kernel="linear", probability=True, random_state=42)
        metrics = {"accuracy": [], "roc_auc": [], "f1": [],"precision": [], "recall": []}

        for train_idx, test_idx in cv.split(X_mel, y_mel):
            X_train, X_test = X_mel[train_idx], X_mel[test_idx]
            y_train, y_test = y_mel[train_idx], y_mel[test_idx]
            svm_model.fit(X_train, y_train)
            y_pred = svm_model.predict(X_test)
            y_proba = svm_model.predict_proba(X_test)[:, 1]
            metrics["accuracy"].append(accuracy_score(y_test, y_pred))
            metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
            metrics["f1"].append(f1_score(y_test, y_pred))
            metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
            metrics["recall"].append(recall_score(y_test, y_pred))

        svm_performance_mel = {metric: {"mean": np.mean(values), "std": np.std(values)} for metric, values in metrics.item
        print("===== SVM Cross-Validation Performance =====")
        for metric, values in svm_performance_mel.items():
            print(f"  {metric.upper()}: {values['mean']:.6f} ± {values['std']:.6f}")
```

```
In [ ]: # Random Forest Model
        cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
        rf_model = RandomForestClassifier(n_estimators=45, max_depth=5, random_state=42)
        metrics = {"accuracy": [], "roc_auc": [], "f1": [],"precision": [], "recall": []}

        for train_idx, test_idx in cv.split(X_mel, y_mel):
            X_train, X_test = X_mel[train_idx], X_mel[test_idx]
            y_train, y_test = y_mel[train_idx], y_mel[test_idx]
            rf_model.fit(X_train, y_train)
            y_pred = rf_model.predict(X_test)
            y_proba = rf_model.predict_proba(X_test)[:, 1]
            metrics["accuracy"].append(accuracy_score(y_test, y_pred))
            metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
            metrics["f1"].append(f1_score(y_test, y_pred))
            metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
            metrics["recall"].append(recall_score(y_test, y_pred))

        rf_performance_mel = {metric: {"mean": np.mean(values), "std": np.std(values)} for metric, values in metrics.items
        print("===== Random Forest Performance =====")
        for metric, values in rf_performance_mel.items():
            print(f"  {metric.upper()}: {values['mean']:.6f} ± {values['std']:.6f}")
```

```
In [ ]: #comparison with two model
        metrics = ["accuracy", "roc_auc", "f1", "precision", "recall"]
        features = ["SVM", "Random Forest"]

        means = np.array([
            [svm_performance_mel[m]["mean"] for m in metrics],
            [rf_performance_mel[m]["mean"] for m in metrics],
        ])
        stds = np.array([
            [svm_performance_mel[m]["std"] for m in metrics],
            [rf_performance_mel[m]["std"] for m in metrics],
        ])

        x = np.arange(len(metrics))
        width = 0.25
        fig, ax = plt.subplots(figsize=(10, 6))
        for i in range(len(features)):
            ax.bar(x + i * width, means[i], width, label=features[i], yerr=stds[i], capsize=5)

        ax.set_xlabel("Metrics")
        ax.set_ylabel("Performance Score")
        ax.set_title("Different model Performance Comparison with Mel spectrum")
        ax.set_xticks(x + width)
        ax.set_xticklabels(metrics)
        ax.legend()
        plt.ylim(0.0, 1.0)
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.show()
```

Supervised learning-Zero crossing rate

```
In [ ]: cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
        svm_model = SVC(kernel="linear", probability=True, random_state=42)
        metrics = {"accuracy": [], "roc_auc": [], "f1": [],"precision": [], "recall": []}
```

```python
for train_idx, test_idx in cv.split(X_zcr, y_zcr):
    X_train, X_test = X_zcr[train_idx], X_zcr[test_idx]
    y_train, y_test = y_zcr[train_idx], y_zcr[test_idx]
    svm_model.fit(X_train, y_train)
    y_pred = svm_model.predict(X_test)
    y_proba = svm_model.predict_proba(X_test)[:, 1]
    metrics["accuracy"].append(accuracy_score(y_test, y_pred))
    metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
    metrics["f1"].append(f1_score(y_test, y_pred))
    metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
    metrics["recall"].append(recall_score(y_test, y_pred))

svm_performance_zcr = {metric: {"mean": np.mean(values), "std": np.std(values)} for metric, values in metrics.item
print("===== SVM Cross-Validation Performance =====")
for metric, values in svm_performance_zcr.items():
    print(f"  {metric.upper()}: {values['mean']:.6f} ± {values['std']:.6f}")
```

In [ ]:

```python
# Random Forest Model
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
rf_model = RandomForestClassifier(n_estimators=45, max_depth=5, random_state=42)
metrics = {"accuracy": [], "roc_auc": [], "f1": [],"precision": [], "recall": []}

for train_idx, test_idx in cv.split(X_zcr, y_zcr):
    X_train, X_test = X_zcr[train_idx], X_zcr[test_idx]
    y_train, y_test = y_zcr[train_idx], y_zcr[test_idx]
    rf_model.fit(X_train, y_train)
    y_pred = rf_model.predict(X_test)
    y_proba = rf_model.predict_proba(X_test)[:, 1]
    metrics["accuracy"].append(accuracy_score(y_test, y_pred))
    metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
    metrics["f1"].append(f1_score(y_test, y_pred))
    metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
    metrics["recall"].append(recall_score(y_test, y_pred))

rf_performance_zcr = {metric: {"mean": np.mean(values), "std": np.std(values)} for metric, values in metrics.items
print("===== Random Forest Performance =====")
for metric, values in rf_performance_zcr.items():
    print(f"  {metric.upper()}: {values['mean']:.6f} ± {values['std']:.6f}")
```

In [ ]:

```python
#comparison with two model
metrics = ["accuracy", "roc_auc", "f1", "precision", "recall"]
features = ["SVM", "Random Forest"]

means = np.array([
    [svm_performance_zcr[m]["mean"] for m in metrics],
    [rf_performance_zcr[m]["mean"] for m in metrics],
])
stds = np.array([
    [svm_performance_zcr[m]["std"] for m in metrics],
    [rf_performance_zcr[m]["std"] for m in metrics],
])

x = np.arange(len(metrics))
width = 0.25
fig, ax = plt.subplots(figsize=(10, 6))
for i in range(len(features)):
    ax.bar(x + i * width, means[i], width, label=features[i], yerr=stds[i], capsize=5)

ax.set_xlabel("Metrics")
ax.set_ylabel("Performance Score")
ax.set_title("Different model Performance Comparison with Zero crossing rate")
ax.set_xticks(x + width)
ax.set_xticklabels(metrics)
ax.legend()
plt.ylim(0.0, 1.0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

Different Feature Extraction comparison

In [ ]:

```python
metrics = ["accuracy", "roc_auc", "f1", "precision", "recall"]
features = ["MFCC", "Mel", "ZCR"]

means = np.array([
    [svm_performance_mfcc[m]["mean"] for m in metrics],
    [svm_performance_mel[m]["mean"] for m in metrics],
    [svm_performance_zcr[m]["mean"] for m in metrics],
])
```

```python
stds = np.array([
    [svm_performance_mfcc[m]["std"] for m in metrics],
    [svm_performance_mel[m]["std"] for m in metrics],
    [svm_performance_zcr[m]["std"] for m in metrics],
])

x = np.arange(len(metrics))
width = 0.25
fig, ax = plt.subplots(figsize=(10, 6))
for i in range(len(features)):
    ax.bar(x + i * width, means[i], width, label=features[i], yerr=stds[i], capsize=5)

ax.set_xlabel("Metrics")
ax.set_ylabel("Performance Score")
ax.set_title("SVM Performance Comparison Across Features")
ax.set_xticks(x + width)
ax.set_xticklabels(metrics)
ax.legend()
plt.ylim(0.0, 1.0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

In [ ]:
```python
metrics = ["accuracy", "roc_auc", "f1", "precision", "recall"]
features = ["MFCC", "Mel", "ZCR"]

means = np.array([
    [rf_performance_mfcc[m]["mean"] for m in metrics],
    [rf_performance_mel[m]["mean"] for m in metrics],
    [rf_performance_zcr[m]["mean"] for m in metrics],
])
stds = np.array([
    [rf_performance_mfcc[m]["std"] for m in metrics],
    [rf_performance_mel[m]["std"] for m in metrics],
    [rf_performance_zcr[m]["std"] for m in metrics],
])


x = np.arange(len(metrics))
width = 0.25
fig, ax = plt.subplots(figsize=(10, 6))
for i in range(len(features)):
    ax.bar(x + i * width, means[i], width, label=features[i], yerr=stds[i], capsize=5)

ax.set_xlabel("Metrics")
ax.set_ylabel("Performance Score")
ax.set_title("Random Forest Performance Comparison Across Features")
ax.set_xticks(x + width)
ax.set_xticklabels(metrics)
ax.legend()
plt.ylim(0.0, 1.0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

Experiment - train set

In [ ]:
```python
splits = [2, 3, 4, 5, 10]
results = {metric: [] for metric in ["accuracy", "roc_auc", "f1", "precision", "recall"]}

for n_splits in splits:
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
    svm_model = SVC(kernel="linear", probability=True, random_state=42)
    metrics = {metric: [] for metric in results.keys()}

    for train_idx, test_idx in cv.split(X_mel, y_mel):
        X_train, X_test = np.array(X_mel)[train_idx], np.array(X_mel)[test_idx]
        y_train, y_test = np.array(y_mel)[train_idx], np.array(y_mel)[test_idx]

        svm_model.fit(X_train, y_train)
        y_pred = svm_model.predict(X_test)
        y_proba = svm_model.predict_proba(X_test)[:, 1]

        metrics["accuracy"].append(accuracy_score(y_test, y_pred))
        metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
        metrics["f1"].append(f1_score(y_test, y_pred))
        metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
        metrics["recall"].append(recall_score(y_test, y_pred))

    for metric in metrics:
        results[metric].append(np.mean(metrics[metric]))
```

```python
plt.figure(figsize=(10, 6))
for metric, values in results.items():
    plt.plot(splits, values, marker="o", label=metric.upper())

plt.xlabel("Number of Splits (n_splits)")
plt.ylabel("Score")
plt.title("SVM Performance Across Different Train/Test Splits")
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:
```python
splits = [2, 3, 4, 5, 10]
results = {metric: [] for metric in ["accuracy", "roc_auc", "f1", "precision", "recall"]}

for n_splits in splits:
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
    rf_model = RandomForestClassifier(n_estimators=45, max_depth=5, random_state=42)
    metrics = {metric: [] for metric in results.keys()}

    for train_idx, test_idx in cv.split(X_mel, y_mel):
        X_train, X_test = np.array(X_mel)[train_idx], np.array(X_mel)[test_idx]
        y_train, y_test = np.array(y_mel)[train_idx], np.array(y_mel)[test_idx]

        rf_model.fit(X_train, y_train)
        y_pred = rf_model.predict(X_test)
        y_proba = rf_model.predict_proba(X_test)[:, 1]

        metrics["accuracy"].append(accuracy_score(y_test, y_pred))
        metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
        metrics["f1"].append(f1_score(y_test, y_pred))
        metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
        metrics["recall"].append(recall_score(y_test, y_pred))

    for metric in metrics:
        results[metric].append(np.mean(metrics[metric]))

plt.figure(figsize=(10, 6))
for metric, values in results.items():
    plt.plot(splits, values, marker="o", label=metric.upper())

plt.xlabel("Number of Splits (n_splits)")
plt.ylabel("Score")
plt.title("Random Forest Performance Across Different Train/Test Splits")
plt.legend()
plt.grid(True)
plt.show()
```

Experiment - imbalance processing

In [18]:
```python
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
svm_model = SVC(kernel="linear", probability=True, random_state=42, class_weight="balanced", C=10)
metrics = {"accuracy": [], "roc_auc": [], "f1": [],"precision": [], "recall": []}

for train_idx, test_idx in cv.split(X_zcr, y_zcr):
    X_train, X_test = X_zcr[train_idx], X_zcr[test_idx]
    y_train, y_test = y_zcr[train_idx], y_zcr[test_idx]
    svm_model.fit(X_train, y_train)
    y_pred = svm_model.predict(X_test)
    y_proba = svm_model.predict_proba(X_test)[:, 1]
    metrics["accuracy"].append(accuracy_score(y_test, y_pred))
    metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
    metrics["f1"].append(f1_score(y_test, y_pred))
    metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
    metrics["recall"].append(recall_score(y_test, y_pred))

svm_performance_zcr_balance = {metric: {"mean": np.mean(values), "std": np.std(values)} for metric, values in metr
print("===== SVM Cross-Validation Performance =====")
for metric, values in svm_performance_zcr_balance.items():
    print(f"  {metric.upper()}: {values['mean']:.6f} ± {values['std']:.6f}")
```

```
===== SVM Cross-Validation Performance =====
  ACCURACY: 0.492308 ± 0.029656
  ROC_AUC: 0.464892 ± 0.045875
  F1: 0.577384 ± 0.016706
  PRECISION: 0.437347 ± 0.018216
  RECALL: 0.850712 ± 0.023535
```

```
In [ ]: metrics = ["accuracy", "roc_auc", "f1", "precision", "recall"]
        features = ["ZCR_pure", "ZCR_balanced"]

        means = np.array([
            [svm_performance_zcr[m]["mean"] for m in metrics],
            [svm_performance_zcr_balance[m]["mean"] for m in metrics],
        ])
        stds = np.array([
            [svm_performance_zcr[m]["std"] for m in metrics],
            [svm_performance_zcr_balance[m]["std"] for m in metrics],
        ])


        x = np.arange(len(metrics))
        width = 0.25
        fig, ax = plt.subplots(figsize=(10, 6))
        for i in range(len(features)):
            ax.bar(x + i * width, means[i], width, label=features[i], yerr=stds[i], capsize=5)

        ax.set_xlabel("Metrics")
        ax.set_ylabel("Performance Score")
        ax.set_title("Random Forest Performance Comparison Across Features")
        ax.set_xticks(x + width)
        ax.set_xticklabels(metrics)
        ax.legend()
        plt.ylim(0.0, 1.0)
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.show()
```

```
In [20]: cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
         rf_model = RandomForestClassifier( n_estimators=45, max_depth=5, class_weight="balanced", random_state=42)
         metrics = {"accuracy": [], "roc_auc": [], "f1": [], "precision": [], "recall": []}

         for train_idx, test_idx in cv.split(X_zcr, y_zcr):
             X_train, X_test = np.array(X_zcr)[train_idx], np.array(X_zcr)[test_idx]
             y_train, y_test = np.array(y_zcr)[train_idx], np.array(y_zcr)[test_idx]

             rf_model.fit(X_train, y_train)
             y_pred = rf_model.predict(X_test)
             y_proba = rf_model.predict_proba(X_test)[:, 1]

             metrics["accuracy"].append(accuracy_score(y_test, y_pred))
             metrics["roc_auc"].append(roc_auc_score(y_test, y_proba))
             metrics["f1"].append(f1_score(y_test, y_pred))
             metrics["precision"].append(precision_score(y_test, y_pred, zero_division=0))
             metrics["recall"].append(recall_score(y_test, y_pred))

         rf_performance_zcr_balanced = {metric: {"mean": np.mean(values), "std": np.std(values)} for metric, values in metr

         print("===== Balanced Random Forest Performance =====")
         for metric, values in rf_performance_zcr_balanced.items():
             print(f"  {metric.upper()}: {values['mean']:.6f} ± {values['std']:.6f}")
```

```
===== Balanced Random Forest Performance =====
  ACCURACY: 0.598741 ± 0.033082
  ROC_AUC: 0.655070 ± 0.016775
  F1: 0.550488 ± 0.038670
  PRECISION: 0.506520 ± 0.034639
  RECALL: 0.604274 ± 0.051538
```

```
In [ ]: metrics = ["accuracy", "roc_auc", "f1", "precision", "recall"]
        features = ["ZCR_pure", "ZCR_balanced"]

        means = np.array([
            [rf_performance_zcr[m]["mean"] for m in metrics],
            [rf_performance_zcr_balanced[m]["mean"] for m in metrics],
        ])
        stds = np.array([
            [rf_performance_zcr[m]["std"] for m in metrics],
            [rf_performance_zcr_balanced[m]["std"] for m in metrics],
        ])


        x = np.arange(len(metrics))
        width = 0.25
        fig, ax = plt.subplots(figsize=(10, 6))
        for i in range(len(features)):
            ax.bar(x + i * width, means[i], width, label=features[i], yerr=stds[i], capsize=5)

        ax.set_xlabel("Metrics")
        ax.set_ylabel("Performance Score")
```

```python
ax.set_title("Random Forest Performance Comparison Across Features")
ax.set_xticks(x + width)
ax.set_xticklabels(metrics)
ax.legend()
plt.ylim(0.0, 1.0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

Unsupervised learning

In [22]:
```python
def kmeans_clustering_PCA(X, y, feature_name):
    kmeans = KMeans(n_clusters=2, random_state=42, n_init=5)
    clusters = kmeans.fit_predict(X)

    acc1 = accuracy_score(y, clusters)
    acc2 = accuracy_score(y, 1 - clusters)
    best_acc = max(acc1, acc2)
    print(f"K-Means Clustering Accuracy ({feature_name}): {best_acc:.4f}")

    #PCA Reduce dimensionality
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)

    fig, axes = plt.subplots(1, 2, figsize=(12, 5))
    # Original labels
    scatter1 = axes[0].scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap="viridis", alpha=0.7)
    axes[0].set_xlabel("PCA Original Component 1")
    axes[0].set_ylabel("PCA Original Component 2")
    axes[0].set_title(f"Original Data Distribution ({feature_name})")
    fig.colorbar(scatter1, ax=axes[0], label="True Label")

    # K-Means clustering results
    scatter2 = axes[1].scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap="coolwarm", alpha=0.7)
    axes[1].set_xlabel("PCA K-means Component 1")
    axes[1].set_ylabel("PCA K-means Component 2")
    axes[1].set_title(f"K-Means Clustering ({feature_name})")
    fig.colorbar(scatter2, ax=axes[1], label="Cluster Label")
    plt.tight_layout()
    plt.show()

    return best_acc
```

In [23]:
```python
def kmeans_clustering_ZCR(X, y):

    kmeans = KMeans(n_clusters=2, random_state=42, n_init=5)
    clusters = kmeans.fit_predict(X)

    acc1 = accuracy_score(y, clusters)
    acc2 = accuracy_score(y, 1 - clusters)
    best_acc = max(acc1, acc2)
    print(f"K-Means Clustering Accuracy (ZCR): {best_acc:.4f}")

    fig, axes = plt.subplots(1, 2, figsize=(12, 5))
    axes[0].scatter(X, np.zeros_like(X), c=y, cmap="viridis", alpha=0.7)
    axes[0].set_xlabel("Zero-Crossing Rate (ZCR)")
    axes[0].set_yticks([])
    axes[0].set_title("Original Labels (ZCR)")

    axes[1].scatter(X, np.zeros_like(X), c=clusters, cmap="coolwarm", alpha=0.7)
    axes[1].set_xlabel("Zero-Crossing Rate (ZCR)")
    axes[1].set_yticks([])
    axes[1].set_title("K-Means Clustering (ZCR)")

    plt.tight_layout()
    plt.show()

    return best_acc
```

In [ ]:
```python
acc_mfcc = kmeans_clustering_PCA(X_mfcc, y_mfcc, "MFCC")
acc_mel = kmeans_clustering_PCA(X_mel, y_mel, "Mel Spectrogram")
acc_zcr = kmeans_clustering_ZCR(X_zcr, y_zcr)
print("\nFinal Comparison of K-Means Clustering Accuracy:")
print(f"MFCC Accuracy: {acc_mfcc:.4f}")
print(f"Mel Spectrogram Accuracy: {acc_mel:.4f}")
print(f"ZCR Accuracy: {acc_zcr:.4f}")
```

In [ ]: