Improvements in Audio Classification with Machine Learning

Contents

1 Feature Engineering: Enhancing Audio Representation

Audio signals often contain high-dimensional data that can be challenging to model directly. We will use **Principal Component Analysis (PCA)** to reduce dimensionality and focus on key features.

Mathematical Insight

PCA projects data into a new space with linearly uncorrelated features (principal components), retaining the maximum variance. For data X (dimensions $n \times d$), PCA transformation finds a new representation X' where:

$$X' = XW$$

where W is a matrix of eigenvectors associated with the largest eigenvalues of the covariance matrix of X.

Code Implementation

```
from sklearn.decomposition import PCA

# Initialize PCA to retain 95% variance
pca = PCA(n_components=0.95)
train_features = pca.fit_transform(embeddings.train)
test_features = pca.transform(embeddings.test)

print(f"Original␣dimensions:␣{embeddings.train.shape[1]},␣Reduced:␣{
    train_features.shape[1]}")
```

2 Ensemble Modeling: Combining Classifier Strengths

Mathematical Insight

Stacking: A meta-learner combines predictions p_1, p_2, p_3 of base models M_1, M_2, M_3:

$$y' = M_{\text{meta}}(p_1, p_2, p_3)$$

Voting: In soft voting, final prediction $\hat{y}$ is:

$$\hat{y} = \arg\max_y \sum_{i=1}^{n} w_i P(y|M_i)$$

where w_i is the weight for model M_i.

Code Implementation

```
from sklearn.ensemble import StackingClassifier, VotingClassifier

# Define base classifiers
base_estimators = [
    ('rf', RandomForestClassifier(n_estimators=100)),
    ('svc', SVC(probability=True)),
    ('lr', LogisticRegression(max_iter=1000))
]

# Stacking and Voting Classifiers
stack_clf = StackingClassifier(estimators=base_estimators, final_estimator
   =LogisticRegression())
vote_clf = VotingClassifier(estimators=base_estimators, voting='soft')

# Fit and evaluate
stack_clf.fit(train_features, labels_train)
vote_clf.fit(train_features, labels_train)
```

3 Hyperparameter Optimization

Using **RandomizedSearchCV** efficiently tunes hyperparameters, sampling a random subset of all combinations.

Code Implementation

```
from sklearn.model_selection import RandomizedSearchCV

param_grid = {
    'rf__n_estimators': [50, 100, 200],
    'svc__C': [0.1, 1, 10],
    'lr__C': [0.1, 1, 10]
}

search = RandomizedSearchCV(vote_clf, param_grid, n_iter=10, cv=5, scoring
   ='accuracy', n_jobs=-1)
search.fit(train_features, labels_train)
```

4 Enhanced Evaluation Metrics

Mathematical Formulation

Precision:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Code Implementation

```
from sklearn.metrics import f1_score, precision_score, recall_score

# Additional evaluation metrics
test_precision = precision_score(all_data.test.labels, y_pred, average='
   weighted')
test_recall = recall_score(all_data.test.labels, y_pred, average='weighted
   ')
test_f1 = f1_score(all_data.test.labels, y_pred, average='weighted')

print(f"Test␣Precision:␣{test_precision:.2f},␣Test␣Recall:␣{test_recall:.2
   f},␣Test␣F1␣Score:␣{test_f1:.2f}")
```

5 Model Persistence

```
import joblib

# Save model
joblib.dump(best_model, 'best_audio_classifier.pkl')

# Load model later
loaded_model = joblib.load('best_audio_classifier.pkl')
```