Lab1 Report for Modeling and Learning form Data

ZHANG Yifei

October 1, 2023

Abstract

This report employs various machine learning methods for classification training and evaluation using a supplied music dataset, and conducts classification on a test set of 200 songs. Standard preprocessing techniques such as normalization are applied to the dataset, and the sklearn toolkit functions are utilized for classification training, with model performance evaluated using k-fold cross-validation. Additionally, this study introduces advanced preprocessing techniques and tunes model parameters to enhance accuracy. Deep learning methods are also employed for training and prediction, and a comparison is drawn with classical machine learning approaches. Finally, the predictive results are uploaded to a website and compared with the accuracy of other classmates' models.

1 Data Processing

1.1 Loading of datasets and test sets

Read the provided dataset and test set using functions from the Pandas package:

```
def load_data(self):
    self.data = pd.read_csv(self.dataset_path)
    self.test_data = pd.read_csv(self.test_data_path)
```

The dataset is known to have a size of [750x14], where each row represents data for an individual song. The first 13 columns correspond to features, while the last column represents the labels. Thus, it is necessary to partition the dataset. Firstly, operations are performed on columns to separate features from labels. Subsequently, operations are conducted on rows to randomly split 20% of the data as the test set, with the remainder forming the training set.

```
# data split
X = self.data.iloc[:, :-1].values
y = self.data.iloc[:, -1].values
self.feature_data = X
self.label_data = y

X_train, X_test, y_train, y_test = train_test_split(filtered_X, filtered_y, ...
    test_size=test_size, random_state=1234)
```

1.2 Data Preprocessing

Data preprocessing is an indispensable component of the machine learning pipeline. It serves to prepare clean and suitable data, providing improved inputs to models, thereby enhancing both model performance and interpretability. Through data preprocessing, issues during model training can be mitigated, ultimately increasing the likelihood of success in machine learning projects.

1.2.1 Data normalizing

Due to the varying scales of each feature in the dataset, normalization is necessary. This aids in mapping the values of different features to a similar scale, thereby enhancing the performance of machine learning models. To achieve this, a data preprocessing tool called MinMaxScaler from Scikit-Learn is employed to scale the features to the [0, 1] range.

```
scaler = MinMaxScaler()
filtered_X = scaler.fit_transform(filtered_X)
```

1.2.2 Singular Value Decomposition

This study employs Singular Value Decomposition (SVD) for further data processing. In certain cases, datasets may contain noise or outliers, which can have adverse effects on the model. SVD can be utilized to detect and remove noise or outliers from the data. Typically, outliers correspond to smaller singular values, and they can be eliminated by excluding the feature vectors associated with these smaller singular values. By sorting the singular values and selecting the top few rows, the feature matrix (dataset) is restructured.

```
def SVD(self, X, N=11):
    U, S, VT = np.linalg.svd(X)
    # Reconstruct the data to keep only the first N singular values
    reconstructed_data = np.dot(U[:, :N], np.dot(np.diag(S[:N]), VT[:N, :]))
    return reconstructed_data
```

1.2.3 Outlier Detection and Removal

Outlier Detection: The process of identifying constant or outlier values in a dataset that deviate significantly from the majority of samples. Used to perform data cleansing, aiming to eliminate noise and anomalies, thereby enhancing the performance of machine learning models.

Outlier Removal: Building upon outlier detection, this involves removing the identified anomalies from the dataset.

```
def outlier_detection(self):
        model = IsolationForest(contamination=0.02)

        # Outlier detection using the fit_predict method
        outliers = model.fit_predict(self.feature_data)

        # Find the index of the samples labeled as outliers
        outlier_indices = np.where(outliers == -1)

        # Remove Outlier
        filtered_X = np.delete(self.feature_data, outlier_indices, axis=0)
        filtered_y = np.delete(self.label_data, outlier_indices, axis=0)

        return filtered_X, filtered_y
```

2 Machine Learning Methods

2.1 K-Nearest Neighbors

K-Nearest Neighbors, abbreviated as KNN, is a supervised learning algorithm used for classification and regression tasks. Its fundamental concept is as follows: if a sample's majority among its K nearest

neighbors in the feature space belongs to a certain category, then that sample is also categorized into that class. In other words, classification is performed using a voting mechanism.

The K-Nearest Neighbors (KNN) algorithm is typically employed in practical applications for small-sized datasets, situations where samples are uniformly distributed, and scenarios that demand strong interpretability.

```
self.model = KNeighborsClassifier(n_neighbors=self.k, weights='distance')
```

By selecting an appropriate value for k, and using k-fold cross-validation to evaluate the model, obtain the training results as shown in Figure 1.

```
准确度: 0.8299319727891157
K折准确度: [0.74666667 0.81333333 0.82666667 0.82666667 0.73333333 0.81333333
 0.77333333 0.86666667 0.74666667 0.78666667]
Predicted Labels: [1 0 0 1 0 0 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0 1
```

Figure 1: KNN Result

Utilizing PCA to reduce features to three dimensions, we observe the distribution of predicted result labels in a three-dimensional space, as shown in the Figure 2.

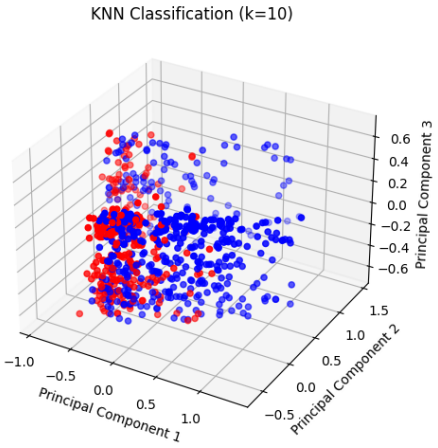


Figure 2: KNN Figure

Utilizing SVD for further data preprocessing and retraining, the results are obtained as depicted in the Figure 3 and Figure 4.

2.2 Random Forest

Random Forest is an ensemble learning algorithm used for addressing classification and regression problems. It enhances model performance and generalization by combining multiple decision trees.

Initially, a certain number of samples (with replacement) and a specific number of features (usually the square root of the total features) are randomly selected from the original training dataset to construct a decision tree. Typically, the CART (Classification and Regression Trees) algorithm is used for this purpose.

Subsequently, the above step is repeated to build multiple decision trees, forming a Random Forest. For classification tasks, the majority voting method is employed to determine the class of a sample.

```
self.model = RandomForestClassifier(n_estimators=95, oob_score=True, random_state=1234)
```

```
准确度:  0.8053691275167785
K折准确度: [0.73333333 0.81333333 0.76       0.82666667 0.76       0.84
 0.78666667 0.85333333 0.74666667 0.78666667]
Predicted Labels: [000100011010010011110011000010100110000001010101100110
```

Figure 3: SVD + KNN Result

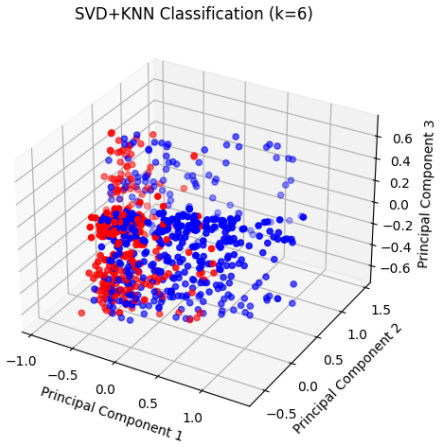


Figure 4: SVD + KNN Figure

The most crucial parameters for Random Forest in scikit-learn are 'n_estimators,' which signifies the number of trees in the forest. Typically, increasing the number of trees can enhance performance but also escalates computational costs. Additionally, 'random_state' (default=None) is a parameter that represents the seed for the random number generator, serving to control the randomness in each training run, ensuring reproducible results.
By fine-tuning the parameters, obtained the training results as shown in the Figure 5.

```
准确度:  0.8707482993197279
K折准确度: [0.8        0.82666667 0.90666667 0.8        0.84       0.84
 0.8        0.92       0.76       0.88       ]
Predicted Labels: [010101010011011010111010001000100111010101011101100011
```

Figure 5: Random Forest Result

Utilizing PCA to reduce features to three dimensions, we observe the distribution of predicted result labels in a three-dimensional space, as depicted in the Figure 6.

Utilizing SVD for further data preprocessing and retraining, the results are obtained as depicted in the Figure 7 and Figure 8.

2.3 Logistic Regression

Logistic Regression is a classical machine learning algorithm primarily used for binary classification tasks. It often serves as one of the benchmark algorithms in machine learning and statistical analysis, particularly well-suited for handling linearly separable problems and high-dimensional data. Logistic Regression typically delivers robust performance when data relationships are approximately linear.
Utilizing k-fold cross validation to evaluate the model, obtained the training results as shown in the 9.
Utilizing PCA to reduce features to three dimensions, we observe the distribution of predicted result labels in a three-dimensional space, as depicted in the 10.
Utilizing SVD for further data preprocessing and retraining, the results are obtained as depicted in the Figure 11 and Figure 12.

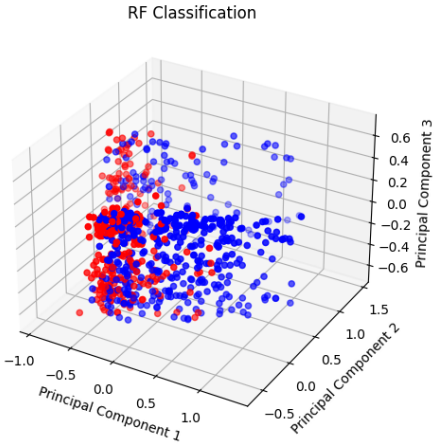


Figure 6: Random Forest Figure

```
准确度:  0.891156462585034
K折准确度: [0.8        0.82666667 0.88       0.8        0.81333333 0.90666667
 0.8        0.88       0.76       0.84       ]
Predicted Labels: [000100010000010010100010000000000010000101010001100010011000

Process finished with exit code 0
```

Figure 7: SVD + Random Forest Result

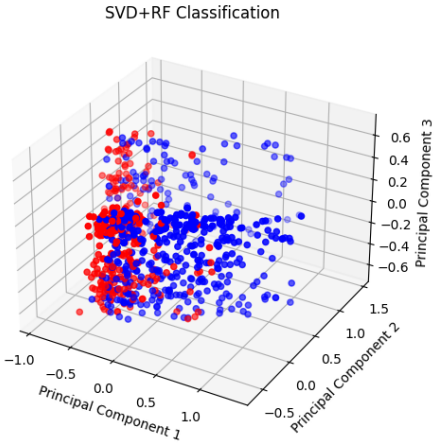


Figure 8: SVD + Random Forest Figure

```
C:\Users\30416\anaconda3\envs\python37\python.exe C:/zhangyifei/study/Lund/2023
准确度:  0.8231292517006803
K折准确度: [0.81333333 0.78666667 0.85333333 0.78666667 0.8        0.85333333
 0.77333333 0.84       0.72       0.88       ]
Predicted Labels: [0 1 0 1 0 1 1 1 0 0 1 1 0 1 1 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0
```

Figure 9: Logistic Regression Result

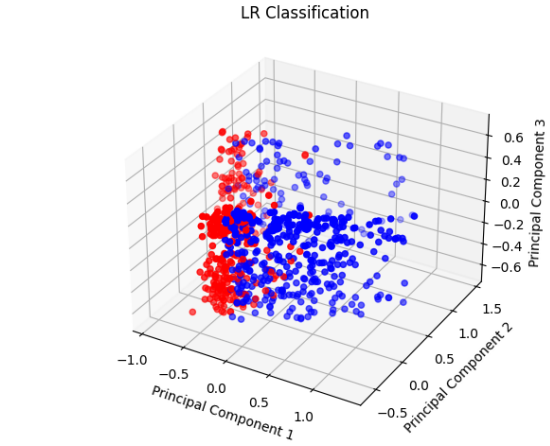


Figure 10: Logistic Regression Figure

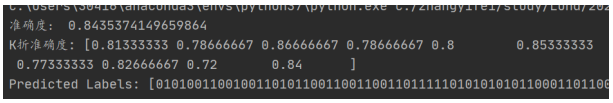


Figure 11: SVD + Logistic Regression Result

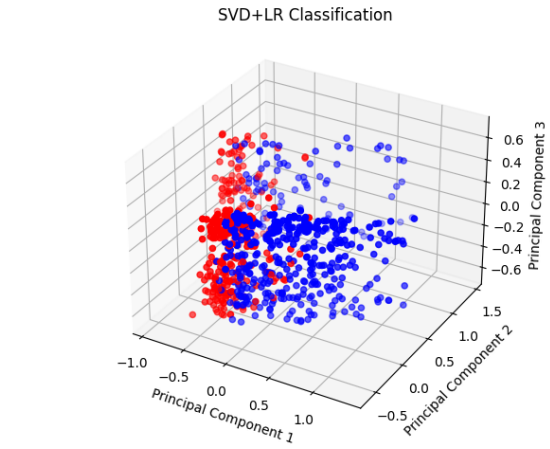


Figure 12: SVD + Logistic Regression Figure

2.4 Voting

Voting is an ensemble learning technique that aggregates predictions from multiple base classifiers. It combines their individual predictions either through majority voting or weighted voting to determine the final outcome of the voting classifier.
Here are the general codes for implementing a voting method for a variety of machine learning algorithms using Voting Classifier:

```
clf1 = SVC(decision_function_shape='ovr', kernel='rbf', probability=True)
clf2 = RandomForestClassifier(n_estimators=95, oob_score=True,
                              random_state=1234)
clf3 = MLPClassifier(hidden_layer_sizes=(256, 128, 64, 32), ...
                                        activation='relu', max_iter=2000)
eclf = VotingClassifier(
            estimators=[('svm', clf1), ('rf', clf2), ('mlp', clf3)],
            voting='hard')
```

The code indicates that Support Vector Classifier (SVC), Random Forest Classifier, and Multi-Layer Perceptron (MLP) have been selected as the fundamental machine learning methods.
By fine-tuning the parameters, obtained the training results as shown in the Figure 13.

```
Accuracy: 0.82 (+/- 0.03) [SVC]
Accuracy: 0.83 (+/- 0.05) [Random Forest]
Accuracy: 0.81 (+/- 0.02) [MLP]
Accuracy: 0.84 (+/- 0.05) [Ensemble]
Predicted Labels: [000101110011011010110011
```

Figure 13: Voting Result

Utilizing PCA to reduce features to three dimensions, we observe the distribution of predicted result labels in a three-dimensional space, as depicted in the Figure 14.

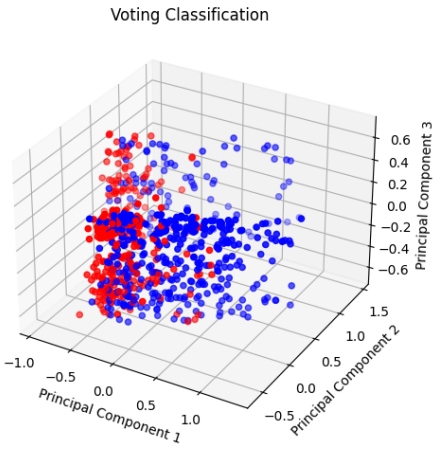


Figure 14: Voting Figure

Utilizing SVD for further data preprocessing and retraining, the results are obtained as depicted in the Figure 15 and Figure 16.

3 Deep Learning Method

he advantages of neural networks lie in their ability to learn and represent complex nonlinear relationships, making them excel at handling intricate data patterns and features. Traditional machine

```
Accuracy: 0.84 (+/- 0.04) [SVC]
Accuracy: 0.88 (+/- 0.05) [Random Forest]
Accuracy: 0.82 (+/- 0.09) [MLP]
Accuracy: 0.84 (+/- 0.05) [Ensemble]
Predicted Labels: [000101110011011010110011000000
```

Figure 15: SVD + Voting Result

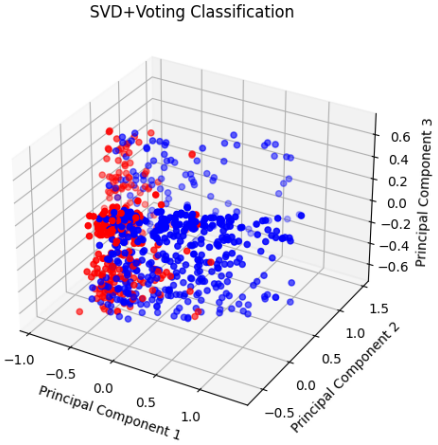


Figure 16: SVD + Voting Figure

learning algorithms like logistic regression and support vector machines are typically linear models, making it challenging for them to capture nonlinear relationships. Properly regularized neural networks often exhibit strong generalization capabilities, delivering good performance on unseen data. Binary classification methods based on fully connected neural networks demonstrate outstanding performance in tackling complex problems and large-scale datasets but demand more data and computational resources, often entailing higher computational complexity and training time.

In contrast, traditional machine learning algorithms are often easier to interpret and understand, providing feature importance analysis. In comparison, neural networks are often considered as 'black-box' models, with their internal decision processes being more challenging to explain.
Below are the steps involved in training a fully connected neural network.
Firstly, the dataset is loaded and preprocessed, followed by randomizing the dataset.

```
# Step 1: Load and Preprocess Data
data = pd.read_csv('training_data.csv')
test_data = pd.read_csv('songs_to_classify.csv')

# data = data.abs()
X = data.iloc[:, :-1].values  # Features (all columns except the last one)
Y = data.iloc[:, -1].values  # Labels (the last column)


batchsz = 32

data = data.abs()
np.random.shuffle(X)
np.random.shuffle(Y)


# Split the data into a training set and a test set
x, x_val, y, y_val = train_test_split(X, Y, test_size=0.2, random_state=42)



scaler = MinMaxScaler()
```

```
x = scaler.fit_transform(x)
x_val = scaler.fit_transform(x_val)

x= SVD(x)
x_val = SVD(x_val)


db = tf.data.Dataset.from_tensor_slices((x, y))
db = db.map(preprocess).shuffle(800).batch(batchsz)
ds_val = tf.data.Dataset.from_tensor_slices((x_val, y_val))
ds_val = ds_val.map(preprocess).batch(batchsz)
```

Next, build the neural network model, define the loss function, and select the optimizer.

```
# Step 2: Build the Neural Network Model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),

    tf.keras.layers.Dense(1, activation='sigmoid')
])
# Step 3: Compile the Model
model.compile(optimizer=optimizers.Adam(learning_rate=0.001),
              # loss=tf.losses.CategoricalCrossentropy(from_logits=True),
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.build(input_shape=(None, x.shape[1]))
model.summary()
```

Finally, the model is trained and tested, and the model weights are saved.

```
# Step 4: Train the Model
model.fit(db, epochs=180, validation_data=ds_val)  # Adjust epochs and batch size as needed

# Step 5: Evaluate the Model
test_loss, test_accuracy = model.evaluate(ds_val)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')

# Step 6:   Save the Model
model.save('my_model5.h5')
```

The structure and training result of the neural network model depicted in Figure 17 and Figure 18.

4 Competition Result

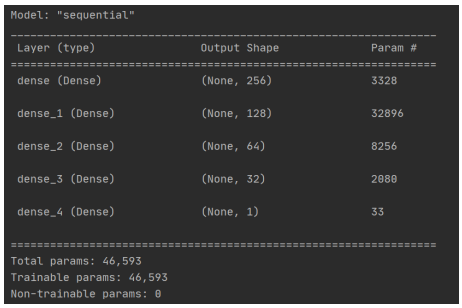
The current position is ranked second on the leaderboard (as of September 30th), group 114 .

5 Conclusion

After testing and analyzing the models, this study draws the following conclusions:

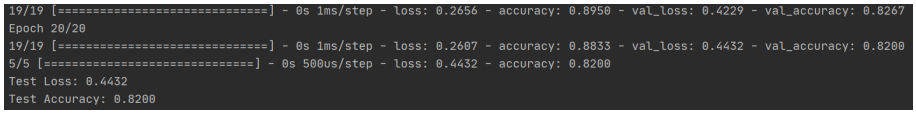
Among traditional machine learning methods, K-Nearest Neighbors (KNN) and Random Forest perform well on the test set, achieving an accuracy of up to 85% after SVD preprocessing. In contrast, Logistic Regression exhibits lower classification accuracy, reaching around 80%. Voting demonstrates higher accuracy, at approximately 83%, but it comes with longer training times.

On the other hand, deep learning methods can easily achieve 100% accuracy on the training set, but they are prone to overfitting, and their classification performance decreases after SVD processing.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               3328

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 1)                 33

=================================================================
Total params: 46,593
Trainable params: 46,593
Non-trainable params: 0
```

Figure 17: Full Connect Model Structure



```
19/19 [==============================] - 0s 1ms/step - loss: 0.2656 - accuracy: 0.8950 - val_loss: 0.4229 - val_accuracy: 0.8267
Epoch 20/20
19/19 [==============================] - 0s 1ms/step - loss: 0.2607 - accuracy: 0.8833 - val_loss: 0.4432 - val_accuracy: 0.8200
5/5 [==============================] - 0s 500us/step - loss: 0.4432 - accuracy: 0.8200
Test Loss: 0.4432
Test Accuracy: 0.8200
```

Figure 18: Full Connect Model Training Result

Position	Group	Performance	Comment (by the group)	Date
1	25	0.825	last dance	2023-09-27
2	113	0.82	Voting test	2023-09-23
2	114	0.82	非常好数据，爱来自资器。	2023-09-30
3	65	0.815	Test submission	2023-09-25
3	23	0.815	Kiktor v2	2023-09-25
3	80	0.815	Day 2: Communism	2023-09-27

Figure 19: Competition Rank

Considering the comprehensive assessment of the test results mentioned above, this study ultimately selects Random Forest.