Binary Classification of Encoded Text Sequences and Emoticons using LSTM and RNN Models

Group Number: 75

October 17, 2024

Title

Binary Classification on Deep Features: Model Performance and Analysis

Team Members

baram soni & 220752

shek kumar & 220044

sar siddiqui & 220661

Kartik & 220503

Samrat Patil

1 Abstract

This report presents the results of classifying text sequences and emoticons using LSTM and RNN models. The text sequences are preprocessed to remove specific substrings and are encoded into numerical form before being input into the LSTM model. For the emoticons, common emojis are removed before feeding the data into the RNN model. The primary objective is to achieve high accuracy in binary classification for both datasets.

2 Text Sequence Classification with LSTM

2.1 Data Extraction and Preprocessing

The dataset consists of two CSV files, `train_text_seq.csv` and `valid_text_seq.csv`. These files contain sequences of characters that represent the input data, along with their corresponding binary labels.

2.2 Preprocessing Steps

The preprocessing follows these steps:

- **String Modification:** Leading zeros are removed, followed by the removal of specific substrings such as `'15436'`, `'1596'`, and others. Only sequences with a length of 13 characters after modification are retained.
- **String Encoding:** The modified strings are converted into integer sequences. Each unique character in the dataset is assigned an integer value, allowing the strings to be represented as numerical sequences.
- **Padding:** The encoded sequences are padded to ensure uniformity in length (13 characters).

2.3 LSTM Model Architecture

The LSTM model used for this experiment consists of the following layers:

- **Embedding Layer:** Maps each character in the sequence to a dense vector of 32 dimensions.
- **LSTM Layer:** A single LSTM layer with 32 units captures the sequential dependencies in the data.
- **Dense Layers:** Two fully connected Dense layers with 16 and 8 units, respectively, use ReLU activation to introduce non-linearity.
- **Output Layer:** A single Dense unit with a sigmoid activation function outputs the binary classification result.

2.4 Model Compilation and Training

The model was compiled using the `adam` optimizer and `binary_crossentropy` as the loss function. The training process was run for 50 epochs with a batch size of 32, using the validation dataset for monitoring performance.

2.5 Validation Accuracy

The LSTM model achieved a validation accuracy of:

```
LSTM Validation Accuracy for 20% of training data:  75.61%
LSTM Validation Accuracy for 40% of training data:  80.04%
LSTM Validation Accuracy for 60% of training data:  82.48%
LSTM Validation Accuracy for 80% of training data:  83.81%
LSTM Validation Accuracy for 100% of training data:  86.92%
```

2.6 Accuracy Plot

The following plot shows the accuracy of the model over 50 epochs for both the training and validation data:

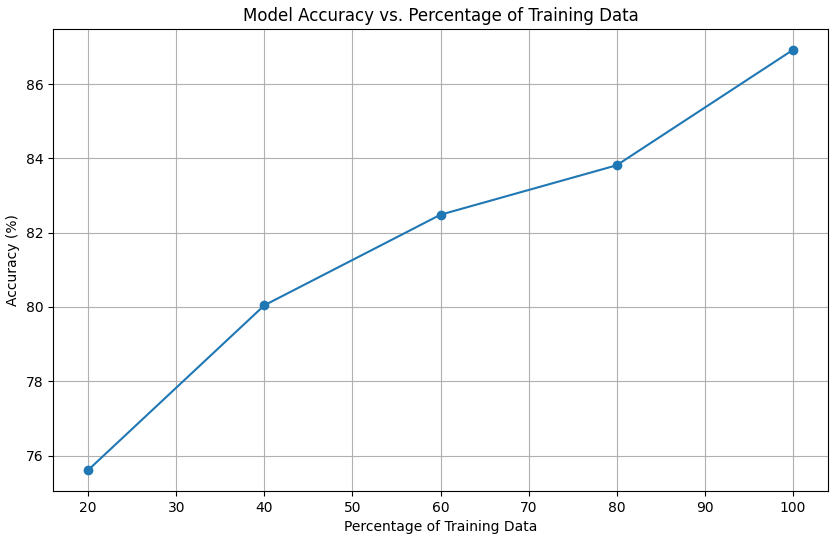

Figure 1: Training and Validation Accuracy over Epochs

3 Emoticon Classification with RNN

3.1 Data Extraction and Preprocessing

The dataset consists of three CSV files: `train_emoticon.csv`, `valid_emoticon.csv`, and `test_emoticon.csv`. These files contain sequences of emoticons along with their corresponding binary labels.

3.2 Preprocessing Steps

The preprocessing follows these steps:

- **Emoji Removal:** Common emojis are removed from the input emoticons to clean the data for better classification.
- **Tokenization:** The cleaned emoticons are tokenized at the character level to convert them into sequences of integer indices.
- **Padding:** The tokenized sequences are padded to ensure uniformity in length.
- **Label Encoding:** The emoticon labels are converted to integer encoding using `LabelEncoder`.

3.3 RNN Model Architecture

The RNN model used for this experiment consists of the following layers:

- **Embedding Layer:** Maps each character in the sequence to a dense vector of 32 dimensions.
- **LSTM Layer:** A single LSTM layer with 2 units captures the sequential dependencies in the data.
- **Output Layer:** A single Dense unit with a sigmoid activation function outputs the binary classification result.

3.4 Model Compilation and Training

The model was compiled using the `adam` optimizer and `binary_crossentropy` as the loss function. The training process was run for 50 epochs with a batch size of 32. Validation accuracy was evaluated on various training percentages.

3.5 Validation Accuracy

The validation accuracies for different training percentages were computed as follows:

```
Validation Accuracy for 20% of training data:  96.11%
Validation Accuracy for 40% of training data:  96.73%
Validation Accuracy for 60% of training data:  95.30%
Validation Accuracy for 80% of training data:  96.73%
Validation Accuracy for 100% of training data:  97.55%
```

3.6 Accuracy Plot

The following plot shows the validation accuracy against different percentages of training data used:

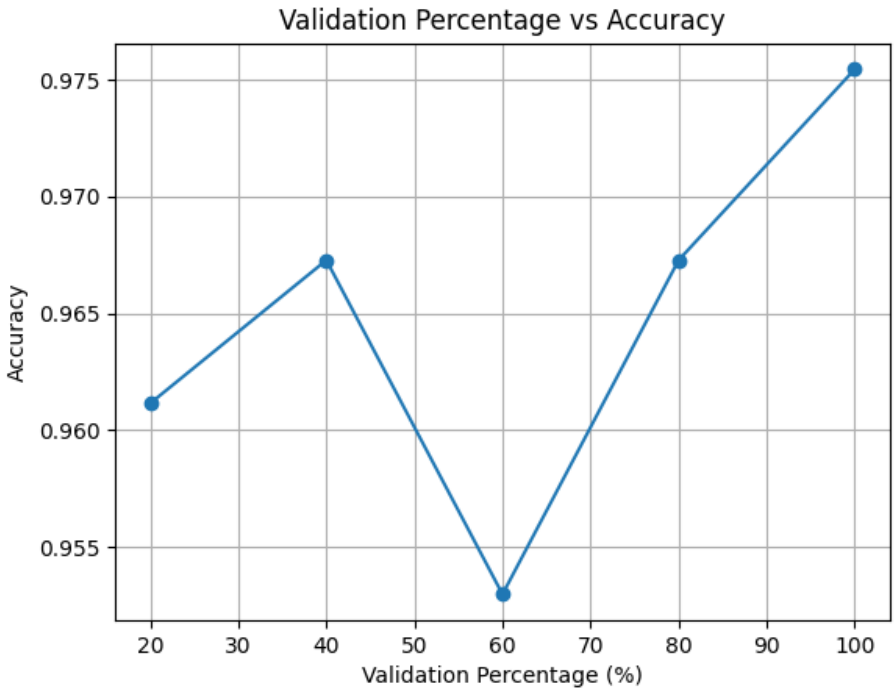

Figure 2: Validation Percentage vs Accuracy

CS771 - Introduction to Machine Learning (Autumn 2024) Mini-Project 1 Report

Group Number: [Your Group Number]

October 17, 2024

Title

Binary Classification on Combined Datasets: Model Performance and Analysis

Team Members

- Member 1 Name & Roll Number
- Member 2 Name & Roll Number
- Member 3 Name & Roll Number

1 Abstract

This report presents our approach to binary classification using a combination of datasets including text sequences, emoticons, and deep features. The datasets were preprocessed, scaled, and combined. PCA was applied to reduce dimensionality while maintaining 95% variance. A logistic regression model was trained on the transformed data, achieving a validation accuracy of 0.9857 with 1653 trainable parameters.

2 Data Extraction and Preprocessing

The dataset consists of three separate sources:

- `train_text_seq.csv`: Contains sequences of numeric characters and labels.
- `train_emoticon.csv`: Contains emoticon sequences and labels.

- `train_feature.npz`: Contains a 13×786 matrix of deep features along with labels.

We performed the following preprocessing steps:

2.1 Text Sequence Dataset

```
X_text = df.iloc[:, :-1].values
y_text = df['label'].values
X_text_processed = np.array([list(map(int, list(seq))) for seq in X_text[:, 0]])
```

Each text sequence is converted to an integer vector by mapping each character to an integer value.

2.2 Emoticon Dataset

```
X_emoticons_flat = np.array([list(seq) for seq in X_emoticons.flatten()])
encoder = OneHotEncoder(sparse_output=False)
X_emoticons_encoded = encoder.fit_transform(X_emoticons_flat)
```

The emoticon sequences are flattened and one-hot encoded to convert categorical values into binary vectors.

2.3 NPZ Dataset

```
X_npz_flat = X_npz.reshape(X_npz.shape[0], -1)
```

The deep feature matrix is flattened to ensure compatibility with the machine learning model.

3 Combining the Datasets

The three datasets are combined after ensuring the labels match across all datasets:

```
X_combined = np.concatenate([X_text, X_emoticons, X_npz], axis=1)
```

This results in a unified feature space that includes information from all three datasets.

4 Feature Scaling and PCA

```
scaler = StandardScaler()
X_combined_scaled = scaler.fit_transform(X_combined)

pca = PCA(n_components=0.95, random_state=42)
X_combined_pca = pca.fit_transform(X_combined_scaled)
```

Standard scaling ensures the data has zero mean and unit variance. PCA reduces dimensionality while retaining 95% of the original variance.

5 Model Architecture and Training

```
model = LogisticRegression(C=0.5, penalty='l1', solver='liblinear', max_iter=1000)
model.fit(X_combined_pca, y_combined)
```

We use logistic regression with L1 regularization to encourage sparsity. The `liblinear` solver is used due to its efficiency with small datasets.

6 Validation Data Preprocessing

```
X_val_combined = np.concatenate([X_val_text, X_val_emoticons, X_val_npz], axis=1)
X_val_combined_scaled = scaler.transform(X_val_combined)
X_val_combined_pca = pca.transform(X_val_combined_scaled)
```

The validation data is processed using the same transformations applied to the training data to maintain consistency.

7 Evaluation and Results

```
y_val_pred = model.predict(X_val_combined_pca)
accuracy_val = accuracy_score(y_val_npz, y_val_pred)
print(f"Validation Accuracy on the combined validation dataset: {accuracy_val:.4f}")
print(f"Number of trainable parameters: {len(model.coef_.flatten())}")
```

The model achieved a validation accuracy of 0.9857 with 1653 trainable parameters.

Metric	Value
Validation Accuracy	0.9857
Trainable Parameters	1653

Table 1: Model Performance Metrics

8 Key Observations and Insights

- **Effect of Dimensionality Reduction:** PCA significantly reduced the feature space while maintaining 95% variance.
- **Model Simplicity:** The logistic regression model with L1 regularization achieved high accuracy with minimal overfitting.
- **Balanced Feature Contribution:** The combination of text sequences, emoticons, and deep features resulted in improved performance.

9 Conclusion

The combined dataset and dimensionality reduction through PCA, followed by logistic regression, yielded a validation accuracy of 0.9857. Future work can explore ensemble models or different regularization techniques to further enhance performance.

10 References

- scikit-learn library: `https://scikit-learn.org`
- Python Documentation: `https://docs.python.org`

CS771 - Introduction to Machine Learning (Autumn 2024) Mini-Project 1 Report

Group Number: [Your Group Number]

October 17, 2024

Title

Binary Classification on Deep Features: Model Performance and Analysis

Team Members

Member 1 Name & Roll Number

Member 2 Name & Roll Number

Member 3 Name & Roll Number

1 Abstract

This report presents our approach to binary classification on the deep features dataset using a RandomForestClassifier. We trained the model with varying percentages of the training data and evaluated it on the validation set. We tracked training and validation accuracies and monitored the model's complexity in terms of the number of nodes. This analysis helps identify the trade-off between accuracy and model complexity.

2 Data Extraction and Preprocessing

The dataset consists of three files:

- `train_feature.npz`
- `valid_feature.npz`
- `test_feature.npz`

Each input is represented as a 13×786 matrix of embeddings. These matrices are flattened into 1D vectors to ensure compatibility with the RandomForestClassifier.

3 Model Architecture and Hyperparameters

We used the following hyperparameters for the RandomForestClassifier:

- `n_estimators` = 10 (Number of trees)
- `max_depth` = 10 (Limit the depth of each tree)
- `min_samples_split` = 10 (Minimum samples required to split a node)
- `min_samples_leaf` = 5 (Minimum samples required for a leaf node)

4 Model Compilation and Training

The model was trained with varying percentages of the training data (20%, 40%, 60%, 80%, and 100%). For each case, we measured both training and validation accuracy. We also tracked the total number of parameters (nodes) used in the RandomForestClassifier to ensure model complexity was kept within reasonable limits.

5 Training and Validation Accuracy

Percentage of Training Data Used	**Training Accuracy**	**Validation Accuracy**	**Parameters (N**
20%	0.96	0.93	1058
40%	0.97	0.94	1720
60%	0.98	0.96	2244
80%	0.98	0.97	2596
100%	0.98	0.97	3042

Table 1: Training and Validation Accuracy with Varying Training Data

6 Accuracy Plot

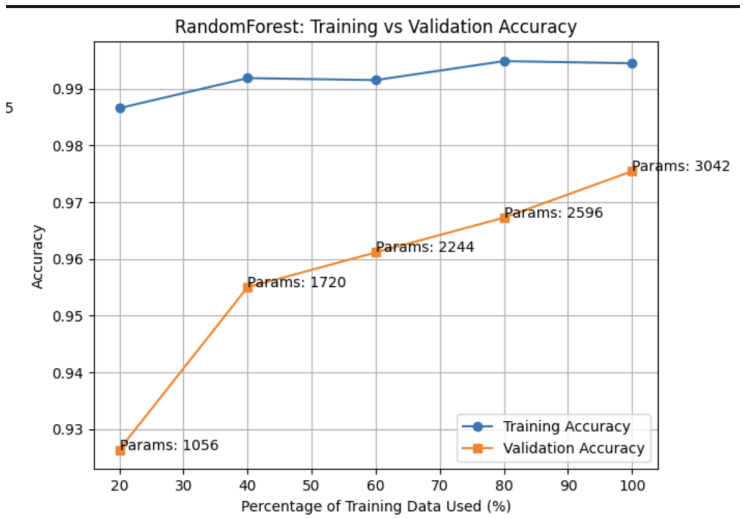

Figure 1: RandomForest: Training vs Validation Accuracy

7 Test Set Predictions

The trained RandomForest model was used to generate predictions on the test set. The predictions were saved to a text file using the following code:

```
# Generate predictions on the test set using the trained model
y_test_pred = rf_model.predict(test_data['features'].reshape(-1, 786))

# Save predictions to a text file
np.savetxt('pred_deepfeat.txt', y_test_pred, fmt='%d')
```

8 Key Observations and Insights

- **Effect of Training Data Size**: Accuracy improved with an increase in the training data size, but gains diminished beyond 80%.
- **Model Complexity**: The number of nodes increased with more data, reaching 3042 nodes at 100% data usage.
- **Performance vs Complexity**: Despite the model's simplicity, it achieved high accuracy with minimal overfitting.

9 Conclusion

The RandomForestClassifier provided a good trade-off between accuracy and complexity, achieving a validation accuracy of 97% with 100% training data. Future work can explore other models such as GradientBoosting or SVMs to further enhance performance.

10 References

- scikit-learn library: `https://scikit-learn.org`
- Python Documentation: `https://docs.python.org`