CS3339: Machine Learning 2024 Fall Project

Haitong Feng-522031910557

***Abstract*—This is a report on a machine learning project to classify high-dimensional sparse vectors. I used three methods: logistic regression, support vector machine (SVM) and neural network for classification and achieved good results.**

***Index Terms*—machine learning, classification, logistic regression, support vector machine, neural network**

I. Introduction

In this project, I completed a classification task with high-dimension sparse data. The data come from an anonymous text classification dataset, in which each text is classified into one of the predefined 20 categories. Pre-extracted features from the dataset are provided, so there is no need to perform feature extraction from raw data.

There are totally 11314 texts for training and another 7532 texts for testing. Each text is represented as a 10000 dim vector. Note that the text features are quite sparse and are in high dimension.

There are many classification methods in machine learning, such as Naive Bayes, SVM, decision tree, K-Nearest Neighbor (KNN), neural network, logistic regression, etc. Based on my personal preferences and proficiency in different models, this project uses three models: logistic regression, SVM and neural network.

II. Overview of Experiment

In this section, I will explain the overall experiment, including: training data analysis, model selection, data processing technique, and model evaluation scheme.

A. Training Data Analysis

The data for this project is provided directly, and I have no knowledge of its data characteristics and prior distribution. Therefore, the analysis of training data is particularly necessary for model selection.

Fig.1 illustrates the label distribution of the training data. It can be found that except for labels 0, 19 and 18, the other labels are relatively evenly distributed. This is friendly for logistic regression.

To obtain the importance of the features, a random forest classifier is used to classify the training data. Table I shows the top 5 most important feature Ordinal Numbers along with their importance. The data show that there are features with significant importance. Of course, this may also be caused by the randomness of the model itself, so it is only for reference.

t-SNE (t-Distributed Stochastic Neighbor Embedding) is a nonlinear dimensionality reduction technique that is primarily used for visualizing high-dimensional data. It can embed high-dimensional data into two-dimensional or three-dimensional spaces while preserving the local structure of the data as much as possible, so that similar data points in the low-dimensional space are still adjacent. The main steps of t-SNE are: for each pair of data points, calculate their similarity in the high-dimensional space and the low-dimensional space, and then minimize the difference in the similarity distribution between the high-dimensional space and the low-dimensional space using gradient descent. Fig.2 shows the t-SNE visualization of the training data in two-dimensional space.

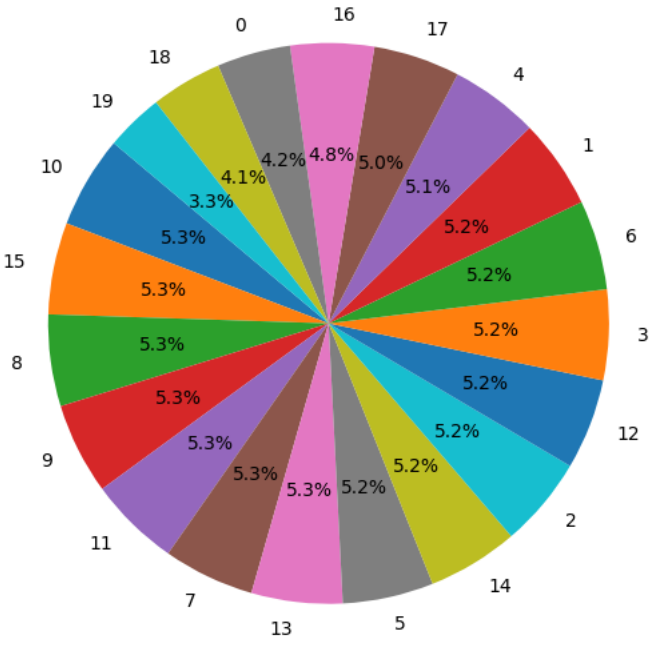


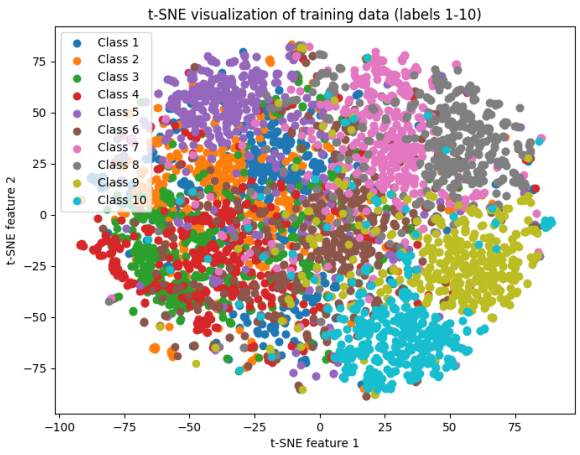
Fig. 1: Label distribution.

TABLE I: Importance Table

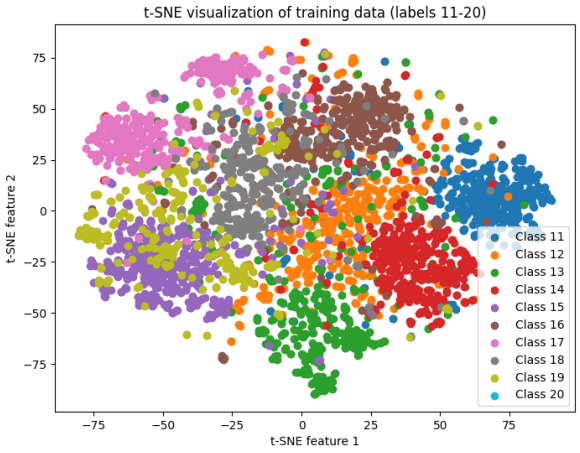
Feature	Importance
7908	0.011217
9767	0.011173
3101	0.009815
1849	0.008212
1481	0.007629

B. Model Selection

Logistic regression is a generalized linear regression analysis model, which is often used in data mining, automatic disease diagnosis, economic forecasting and other fields. Logistic regression is often used for binary classification tasks, but can also be used for multiclass classification tasks. For multiclass classification tasks, there are two common strategies, One-vs-Rest (OvR) and One-vs-One (OvO). The distribution of tags



(a)



(b)

Fig. 2: t-SNE visualization of training data.

in this project is relatively even, and the total number of tags is only 20, so logistic regression model is a good choice.

SVM is a kind of generalized linear classifier that classifies data in a supervised learning way. Its decision boundary is the maximum margin hyperplane solved on the learning samples. Similar to logistic regression, SVM has two strategies for multiclass classification: OvR and OvO. However, SVM can also be used for non-linear classification using kernel methods, which is one of the most common kernel learning methods. Therefore, I chose SVM to classify the data.

Neural network is a machine learning model that simulates the structure and function of the human brain. It consists of a large number of neurons connected by weights. The neural network model has strong nonlinear fitting ability and can be used for both regression and classification tasks. Not only that, the number of neurons in the hidden layer and the correlation between the hidden layers are adjustable parameters, so the neural network is easy to prevent underfitting and overfitting. Therefore, this model is my last choice.

C. Data Processing Technique

The data in this project is sparse and high-dimensional. In order to improve the efficiency of the model, some specific data processing methods should be used to reduce the dimensionality of the original data.

Principal component analysis (PCA) is a statistical method that transforms a set of possibly correlated variables into a set of linearly uncorrelated variables by orthogonal transformation, which is called principal components. The idea of PCA is to map n-dimensional features onto m-dimensions $(m < n)$. These m dimensions are new orthogonal features called principal components. The core is to project the data along the maximum direction, which makes the data easier to distinguish.

Singular value decomposition (SVD) is a matrix factorization method. It is a concept of linear algebra, but it is widely used in machine learning. Singular value decomposition decomposes any given matrix into a product of three matrices: an orthogonal matrix of left singular vectors, a diagonal matrix of singular values, and an orthogonal matrix of right singular vectors. The singular value representations of the data set are arranged according to their importance, and the unimportant eigenvectors are discarded to achieve the purpose of dimensionality reduction, so as to find the principal components in the data.

Since the data itself is a high-dimensional vector, the SVD is less effective at reducing dimensionality. Therefore, I chose PCA as the data processing method. However, in the experiment, training the model with the reduced data did not perform as well as using the original data. This part will be elaborated in III.

D. Model Evaluation Scheme

In machine learning and data science, a model evaluation scheme is a method used to evaluate model performance and select the best model.

A common approach is the Train/Validation/Test Split. It divides the dataset into training set, validation set and test set. The training set is used to train the model, the validation set is used to tune the model parameters, and the test set is used to evaluate the final performance of the model.

Cross-validation is a more robust way to evaluate models, especially with small amounts of data. Common Cross-Validation methods include K-Fold Cross-Validation and Leave-One-Out cross-validation.

According to the experimental requirements, I chose to use cross-validation to evaluate the model's generalization ability. 10-fold cross-validation is more suitable for cases where the data volume is large and high precision is required, so I chose it.

III. Experiment Results and Analysis

A. Logistic Regression

Logistic regression is the first model in this experiment. The associated code is stored in the file **logistic_regression.py**.

Logistic regression in scikit-learn implements binary, OvR, and multinomial logistic regression in the **LogisticRegression** class, with optional L1 and L2 regularization. Logistic regression in scikit-learn uses L2 regularization by default. As an optimization problem, binary logistic regression with L2 penalty minimizes the following cost function:

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(x_i^T w + b)) + 1) \quad (1)$$

Similarly, logistic regression with L1 regularization solves the following optimization problem:

$$\min_{w,b} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(x_i^T w + b)) + 1) \quad (2)$$

The **LogisticRegression** class provides these optimization algorithms: **liblinear**, **newton-cg**, **lbfgs**, **sag**, and **saga**. The **liblinear** applies Coordinate Descent (CD) algorithm and is implemented based on **LIBLINEAR** library, a high-performance C++ library included in scikit-learn. **lbfgs**, **sag**, and **newton-cg** solvers support only L2 penalty terms as well as no penalty terms, and converge faster for certain high-dimensional data. The **sag** solver is based on Stochastic Average Gradient Descent (SAGD). Its performance is faster on large datasets, which are large in size and have many features. The **saga** solver is a variant of sag that supports the non-smooth L1 regularization option.

In this experiment, I trained 6 models using logistic regression. The differences between them are: whether the solver is **saga** or **liblinear**, whether PCA is used for dimensionality reduction, and whether the dimensionality is reduced to 100 or 1000. The accuracy, precision, recall and loss of the model on the validation set are calculated respectively. Its output is shown in Fig.3. The accuracy on the test set is shown in Fig.4. Table II summarizes the above data for comparison.

From the results, we can see that both solvers **saga** and **liblinear** have similar performance. Training the model using the data after PCA dimensionality reduction is significantly less effective than using the source data directly. Although the performance of PCA 1000 is significantly better than that of PCA 100, training the model using PCA 1000 occupies the vast majority of the time in the experiments. The same is true for other classification methods. Therefore, the effect of directly using the source data is optimal in both results and time.

The reason for the difference in solution results may be the difference between the SAGD algorithm and the CD algorithm. CD algorithm has a large amount of calculation and slow convergence speed when dealing with high-dimensional data. Therefore, **liblinear** solver gives slightly worse results than **saga**, probably because **liblinear** requires a higher number of iterations. PCA loses information in the process of dimensionality reduction. Therefore, PCA 100 does not perform well due to too much missing information. On the other hand, it may be that the data after dimensionality reduction contains a large number of floating point numbers, which causes the PCA 1000 training time to be unusually long.

```
SAGA Solver - Accuracy: 0.8899
SAGA Solver - Precision: 0.8925
SAGA Solver - Recall: 0.8874
SAGA Solver - Loss: 0.8930
Liblinear Solver - Accuracy: 0.8849
Liblinear Solver - CPrecision: 0.8874
Liblinear Solver - Recall: 0.8817
Liblinear Solver - Loss: 0.9825
PCA 100 + SAGA Solver - Accuracy: 0.7977
PCA 100 + SAGA Solver - Precision: 0.7983
PCA 100 + SAGA Solver - Recall: 0.7903
PCA 100 + SAGA Solver - Loss: 1.0969
PCA 100 + Liblinear Solver - Accuracy: 0.7922
PCA 100 + Liblinear Solver - Precision: 0.7933
PCA 100 + Liblinear Solver - Recall: 0.7832
PCA 100 + Liblinear Solver - Loss: 1.2219
PCA 1000 + SAGA Solver - Accuracy: 0.8763
PCA 1000 + SAGA Solver - Precision: 0.8793
PCA 1000 + SAGA Solver - Recall: 0.8740
PCA 1000 + SAGA Solver - Loss: 0.8883
PCA 1000 + Liblinear Solver - Accuracy: 0.8726
PCA 1000 + Liblinear Solver - Precision: 0.8753
PCA 1000 + Liblinear Solver - Recall: 0.8692
PCA 1000 + Liblinear Solver - Loss: 0.9941
```

Fig. 3: Logistic regression models training results.

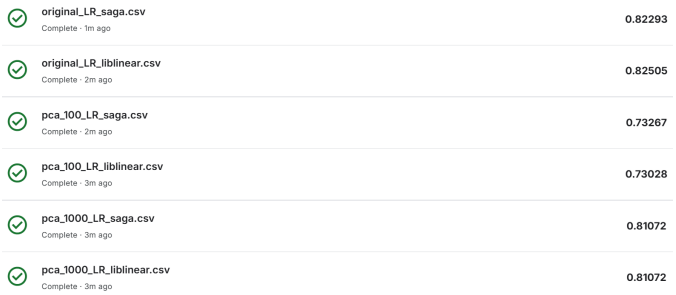


Fig. 4: Logistic regression models test results.

B. Support Vector Machine

SVM is the second model in this experiment. The associated code is stored in the file **SVM.py**.

SVM in scikit-learn uses OvR in **LinearSVC** class for multiclass classification. And in the **SVC** class with linear kernel, OvO can be used for multiclass classification. SVMs use L2 regularization by default. As an optimization problem, binary logistic regression with L2 penalty term minimizes the following cost function:

TABLE II: Logistic Regression Models Experimental Results

Model	Validation Accuracy	Validation Precision	Validation Recall	Validation Loss	Test Accuracy
Saga	0.8899	0.8925	0.8874	0.8930	0.82293
Liblinear	0.8849	0.8874	0.8817	0.9825	0.82505
PCA 100 + Saga	0.7977	0.7983	0.7903	1.0969	0.73267
PCA 100 + Liblinear	0.7922	0.7933	0.7832	1.2219	0.73028
PCA 1000 + Sag	0.8763	0.8793	0.8740	0.8883	0.81072
PCA 1000 + Liblinear	0.8726	0.8753	0.8692	0.9941	0.81072

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \tag{3}$$

$$\text{s.t.} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \quad i = 1, \ldots, n \tag{4}$$

$$\zeta_i \geq 0, \quad i = 1, \ldots, n \tag{5}$$

Where $\phi(x)$ is the feature mapping.

The essence of kernel methods is to map low-dimensional linearly non-separable data into a high-dimensional space, making it linearly separable. Since the data itself is a high-dimensional sparse vector, applying kernel methods would only make the situation more complicated. In this experiment, I trained 3 models using SVM. The differences between them are: whether to use PCA for dimensionality reduction, and whether to reduce to 100 or 1000 dimensions. The accuracy, precision and recall of the model on the validation set are calculated respectively. Its output is shown in Fig.5. The accuracy on the test set is shown in Fig.6. Table III summarizes the above data for comparison.

```
LinearSVC - Original Data - Accuracy: 0.9097
LinearSVC - Original Data - Precision: 0.9107
LinearSVC - Original Data - Recall: 0.9093
LinearSVC - PCA 100 - Accuracy: 0.8047
LinearSVC - PCA 100 - Precision: 0.8000
LinearSVC - PCA 100 - Recall: 0.7965
LinearSVC - PCA 1000 - Accuracy: 0.8924
LinearSVC - PCA 1000 - Precision: 0.8925
LinearSVC - PCA 1000 - Recall: 0.8916
```

Fig. 5: SVM models training results.

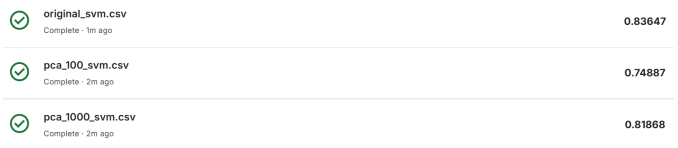


Fig. 6: SVM models test results.

From the results, SVM generally performed slightly better than logistic regression. The original data still performs better than the reduced data. The problem with PCA 1000 might be the lack of features, and the problem with PCA 100 might be that it's not linearly separable. Out of curiosity, I'm going to try applying a kernel to PCA 100. The relevant code is stored in **SVM_PCA_100.py**.

The kernel functions used in this study are polynomial kernel and Radial Basis function (RBF) kernel.

The polynomial kernel is expressed as follows:

$$K(x, x') = (\gamma x^T x' + r)^d \tag{6}$$

Where d is the degree of the polynomial, r is the coefficient, and γ is the kernel coefficient. By default, $r = 0$ and $\gamma = 1/(n\_features * X.var())$. $n\_features$ is the number of features and $X.var()$ is the variance of the data

The RBF kernel is expressed as follows:

$$K(x, x') = \exp(-\gamma \left\| x - x' \right\|_2^2) \tag{7}$$

The accuracy, precision and recall of the model on the validation set are calculated respectively. Its output is shown in Fig.7. The accuracy on the test set is shown in Fig.8. Table IV summarizes the above data for comparison.

```
SVC with Polynomial Kernel (degree=2) - PCA 100 - Accuracy: 0.7330
SVC with Polynomial Kernel (degree=2) - PCA 100 - Precision: 0.8288
SVC with Polynomial Kernel (degree=2) - PCA 100 - Recall: 0.7291
SVC with Polynomial Kernel (degree=3) - PCA 100 - Accuracy: 0.5966
SVC with Polynomial Kernel (degree=3) - PCA 100 - Precision: 0.8347
SVC with Polynomial Kernel (degree=3) - PCA 100 - Recall: 0.5931
SVC with RBF Kernel - PCA 100 - Accuracy: 0.8224
SVC with RBF Kernel - PCA 100 - Precision: 0.8253
SVC with RBF Kernel - PCA 100 - Recall: 0.8201
```

Fig. 7: Kernel methods training results.

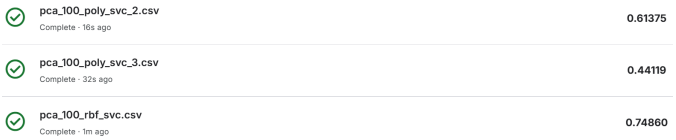


Fig. 8: Kernel methods test results.

From the experimental results, we can see that the RBF kernel has better performance than the polynomial kernel. The accuracy of the polynomial kernel increases with the degree, although it decreases significantly in all other respects. RBF performs better than other kernels on the validation set. However, this does not contribute much to its accuracy improvement on the test set. In summary, it is useless to apply a kernel to the reduced data because too many features are lost and the risk of overfitting after applying the kernel increases significantly.

TABLE III: SVM Models Experimental Results

Model	Validation Accuracy	Validation Precision	Validation Recall	Test Accuracy
Original Data	0.9097	0.9107	0.9093	0.83647
PCA 100	0.8047	0.8000	0.7965	0.74887
PCA 1000	0.8924	0.8925	0.8916	0.81868

TABLE IV: Kernel methods Experimental Results

Kernel	Validation Accuracy	Validation Precision	Validation Recall	Test Accuracy
Linear	0.8047	0.8000	0.7965	0.74887
Polynomial (2)	0.7330	0.8288	0.7291	0.61375
Polynomial (3)	0.5966	0.8347	0.5931	0.44119
RBF	0.8224	0.8253	0.8201	0.74860

C. Neural Network

Neural network is the last model in this experiment. The associated code is stored in the file **neural_network.py**.

The training of the Ann model is very time-consuming, so the key of this experiment is not the comparison between the original data and the dimensionality reduction results of the trained data, but the design of the Ann model. Therefore, I use the raw data as the only training data to compare the training effects of different models. In addition, cross-validation requires training the model repeatedly and is almost as effective as splitting the validation set directly. Therefore, 10-fold cross-validation is not used in this experiment.

The activation functions commonly used in neural networks are sigmoid, Tanh and ReLU. The sigmoid function is shown below:

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{8}$$

The Tanh function is shown below:

$$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{9}$$

The ReLU function is shown below:

$$f(x) = \max(0, x) \tag{10}$$

Tensorflow provides convenient neural network training functions. In this experiment, I used a three-hidden layer neural network with 512, 256 and 128 nodes in each layer. I trained 6 models that differed in the type of activation function and whether or not Dropout was used. The probability of Dropout is set to 0.5. The accuracy, precision, recall and loss of the model on the validation set are calculated respectively. Its output is shown in Fig.9. The accuracy on the test set is shown in Fig.10. Table V summarizes the above data for comparison.

From the results, the final test results of the three activation functions in the fully connected neural network model are similar. After adding Dropout, the test accuracy of ReLU rises significantly. However, the accuracy of Tanh rises slightly and even the accuracy of Sigmoid decreases significantly. It could

```
Validation Accuracy (relu, Dropout=False): 0.8816
Validation Precision (relu, Dropout=False): 0.8828
Validation Recall (relu, Dropout=False): 0.8803
Validation Loss (relu, Dropout=False): 0.8064
```

(a)

```
Validation Accuracy (sigmoid, Dropout=False): 0.8555
Validation Precision (sigmoid, Dropout=False): 0.8545
Validation Recall (sigmoid, Dropout=False): 0.8515
Validation Loss (sigmoid, Dropout=False): 0.7366
```

(b)

```
Validation Accuracy (tanh, Dropout=False): 0.8767
Validation Precision (tanh, Dropout=False): 0.8775
Validation Recall (tanh, Dropout=False): 0.8744
Validation Loss (tanh, Dropout=False): 0.6928
```

(c)

```
Validation Accuracy (relu, Dropout=True): 0.8948
Validation Precision (relu, Dropout=True): 0.8945
Validation Recall (relu, Dropout=True): 0.8933
Validation Loss (relu, Dropout=True): 0.7843
```

(d)

```
Validation Accuracy (sigmoid, Dropout=True): 0.8524
Validation Precision (sigmoid, Dropout=True): 0.8545
Validation Recall (sigmoid, Dropout=True): 0.8503
Validation Loss (sigmoid, Dropout=True): 0.5503
```

(e)

```
Validation Accuracy (tanh, Dropout=True): 0.8829
Validation Precision (tanh, Dropout=True): 0.8817
Validation Recall (tanh, Dropout=True): 0.8803
Validation Loss (tanh, Dropout=True): 0.9519
```

(f)

Fig. 9: Neural network models training results.

TABLE V: Neural Network Models Experimental Results

Model	Validation Accuracy	Validation Precision	Validation Recall	Validation Loss	Test Accuracy
ReLU	0.8816	0.8828	0.8803	0.8064	0.79400
Sigmoid	0.8555	0.8545	0.8515	0.7366	0.78364
Tanh	0.8767	0.8775	0.8744	0.6928	0.78709
ReLU + Dropout	0.8948	0.8945	0.8933	0.7843	0.82134
Sigmoid + Dropout	0.8524	0.8545	0.8503	0.5503	0.76506
Tanh + Dropout	0.8829	0.8817	0.8803	0.9519	0.79400

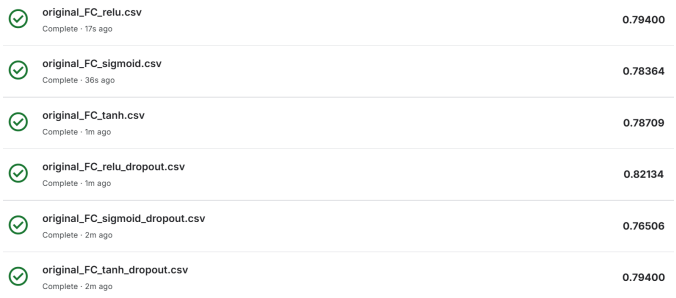
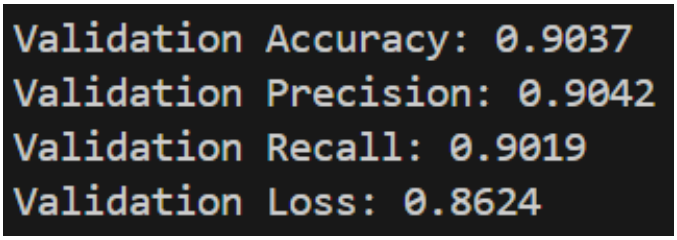


Fig. 10: Neural network models test results.

be that the gradient descent of the Sigmoid function is slow, causing the model not to be fully trained. Adding Dropout only makes underfitting worse. Considering that Dropout gave the best performance for ReLU, I tried to improve it even further.

After repeatedly changing the Dropout probability and the number of epochs, I finally obtained the relatively best neural network model. Its Dropout probability is 0.7 and the number of epochs is 60. This code is stored in **best_neural_network.py**. The final experimental results are shown in Fig.11.



(a)

(b)

Fig. 11: Best neural network models training and test results.

A higher probability of Dropout reduces the risk of overfitting, but makes convergence slower. After the probability increases to 0.8, an extremely large number of epochs is required to train adequately. And it's not a significant improvement in the results.

IV. Conclusion

In this project, I used logistic regression, SVM and neural networks to classify high-dimensional sparse vectors. The experimental results show that the original data is better than the reduced data. The logistic regression model is the fastest to train, but not as accurate as the other two models. The neural network model performs better but takes the longest time to train. The ReLU activation function is the best choice for the neural network model. The Dropout method can improve the performance of the model, but the Dropout probability should not be too high. The SVM model performed the best, but the kernel method did not improve the performance of the model. In the end, my highest test accuracy was 0.83647, which came from the SVM result.

The results indicate that while dimensionality reduction techniques like PCA can help in reducing the computational complexity, they may also lead to a loss of important information, which can negatively impact the model's performance. Therefore, it is crucial to find a balance between dimensionality reduction and retaining essential features.

Furthermore, the choice of activation function and regularization techniques like Dropout play a significant role in the performance of neural networks. The ReLU activation function, combined with an appropriate Dropout rate, can significantly enhance the model's ability to generalize to unseen data. However, it is important to carefully tune these hyperparameters to avoid underfitting or overfitting.

In future work, it would be interesting to explore other advanced techniques such as ensemble methods, which combine multiple models to improve overall performance. Additionally, experimenting with different types of neural network architectures, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), could provide further insights into their applicability to high-dimensional sparse data.

Overall, this project highlights the importance of selecting the right model, preprocessing techniques, and hyperparameters to achieve optimal performance in machine learning tasks. By understanding the strengths and limitations of each approach, we can make more informed decisions and develop more robust models for various applications.