Machine Learning Final Project

110550068 王振倫

Introduction:

In this final project, I primarily utilized K-Nearest Neighbors and Multilayer Perceptron Classifier for classification, analyzing 77 features with the aim of predicting unknown cases. In addition, I also implemented an Naive Bayes Classifier model.

Implementation Details:

Data Preprocessing

1. K-fold
   This project utilizes K-Fold cross-validation, and to avoid potential data leakage issues, I perform preprocessing after the data splits. This prevents the validation data from being influenced by the training data. Although it increases the number of processing steps, it effectively aids in predicting unseen data.
2. Missing values & Standardization

Missing values:

- ➢ For numerical features (F1~F17):
  When dealing with missing values, I use the preceding data for imputation. If the first entry has a missing value, I directly use the median.
- ➢ For binary features (F18~F77):
  I fill them based on the data distribution in that column, probabilistically choosing between 0 or 1. For instance, if the number of 0s is particularly high in that column, there is a higher probability of filling the missing value with 0.

Standardization:

- ➢ For numerical features (F1~F17):
  I apply StandardScaler(), transforming numerical values to a distribution with a mean of 0 and a standard deviation of 1. This helps prevent the scale

from influencing the model's judgment.

- For binary features (F18~F77):
  I utilize LabelEncoder to map each category to an integer value for processing binary features.

Architectures of classifiers

1. KNN:
   Introduction: KNN primarily aims to determine the result for each instance based on the closest neighbors. It is believed that if the features are similar, which means the distance will be close, leading to similar classification results.
   fit(): During the fitting process, we only need to store the training data, without the need for pre-training.
   predict(): After inputting X_train_fold, the predict_proba function is called to obtain the probability of an instance being classified as 1. If this probability is greater than or equal to 0.5, it predicts as 1; otherwise, it predicts as 0.
   predict_proba(): Upon receiving X_train_fold, for each instance, differences are obtained by subtraction, and the Euclidean Distance is used as the distance.Then, it is stored for all instances in X_train. Subsequently, for each instance, its distances to all instances in X_train are sorted using the specified distance metric, and the closest k items are recorded. Finally, for these k items, their classifications are checked, and the probability of predicting 1 is returned.
   distance function: I have experimented with various distance metrics such as Minkowski Distance, Euclidean Distance, Manhattan Distance, etc. Through experimentation, I found that Euclidean Distance yielded the best predictive performance for this dataset.
   Choice of K: In the implementation, I experimented with values of k ranging from 3 to 37. I observed that the average performance was optimal when k was set to 7. Therefore, I ultimately chose k=7 for the model.

2. Multilayer Perceptron Classifier:
   Init(): Besides setting the learning rate = 0.8, it is also necessary to provide initial values for weights and biases.
   fit(): In the fit function, I use One-hot encoding to encode target labels and then

execute both forward_propagation() and backward_propagation() processes for epochs = 300 times.

forward_propagation(): The features of train instances are first inputted. This input is then multiplied by weights, and biases are added. After passing through the sigmoid function, the output of that layer is obtained. Finally, at the output layer, the softmax function is employed to obtain the probability for each classification.

backward_propagation(): The derivative of the sigmoid function, which is output * (1 - output), multiplied by (target - output), yields the delta. Subsequently, the weights and biases are adjusted backward based on this delta and the learning rate.

predict(): The features of instances are used as input and passed into predict_proba.

predict_proba(): After traversing through each layer, the result at the output layer is obtained, and the prediction is made based on the output with the highest probability.

(Additional)

3. Naive Bayes Classifier:

Introduction: The Naive Bayes Classifier predicts the probabilities for specific features by precomputing the conditional probabilities of the training data.

fit(): When X_train_fold is passed, it needs to group the data based on whether the outcome is 1 or 0. Then, it calculates the probabilities of each feature occurring under the condition of outcome being 1 or 0. It further separates features into numerical and binary categories. For numerical features, it computes the mean and standard deviation, while for binary features, it directly calculates the probabilities of values being 1 and 0. Here, to address potential extreme imbalances in features, if the total count for a specific value of a feature is too low, m-smoothing is applied to adjust for such extreme cases.

predict(): After inputting X_train_fold, the predict_proba function is called to obtain the probability of an instance being classified as 1. If this probability is greater than or equal to 0.5, it predicts as 1; otherwise, it predicts as 0.

predict_proba(): Upon receiving X_train_fold, for each instance, assuming independence among features, it calculated the probability of each feature occurrence. For numerical features, the mean and standard deviation calculated during the fit() are used, and the probability is computed using the probability density function. For binary features, the observed probabilities from the fit() are used for calculations. Finally, normalization is applied to ensure that the sum of probabilities for predicting 0 and predicting 1 equals 1. The function then returns the probability of predicting 1.

Discussion

K-fold Result & comparative assessment

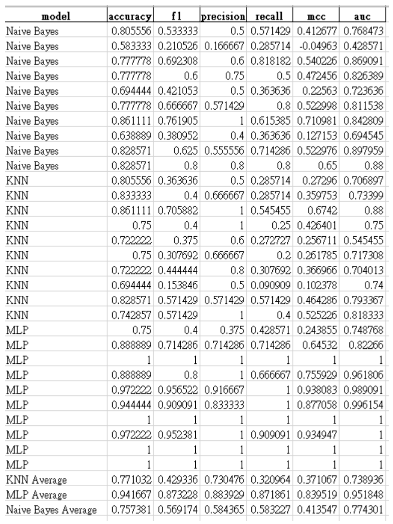
I obtained the result:

While models = {'Naive Bayes': NaiveBayesClassifier(), 'KNN': KNearestNeighbors(), 'MLP': MultilayerPerceptron(77,[32, 24, 8],2)}

I setting the parameters as follows:

For KNN, I used Euclidean distance with k=7.

For MLP, the architecture consists of layers with dimensions (77, [32, 24, 8], 2), and I set the number of epochs to 300 with a learning rate of 0.8.



model	accuracy	f1	precision	recall	mcc	auc
Naive Bayes	0.805556	0.533333	0.5	0.571429	0.412677	0.768473
Naive Bayes	0.583333	0.210526	0.166667	0.285714	-0.04963	0.428571
Naive Bayes	0.777778	0.692308	0.6	0.818182	0.540226	0.869091
Naive Bayes	0.777778	0.6	0.75	0.5	0.472456	0.826389
Naive Bayes	0.694444	0.421053	0.5	0.363636	0.22563	0.723636
Naive Bayes	0.777778	0.666667	0.571429	0.8	0.522998	0.811538
Naive Bayes	0.861111	0.761905	1	0.615385	0.710981	0.842809
Naive Bayes	0.638889	0.380952	0.4	0.363636	0.127153	0.694545
Naive Bayes	0.828571	0.625	0.555556	0.714286	0.522976	0.897959
Naive Bayes	0.828571	0.8	0.8	0.8	0.65	0.88
KNN	0.805556	0.363636	0.5	0.285714	0.27296	0.706897
KNN	0.833333	0.4	0.666667	0.285714	0.359753	0.73399
KNN	0.861111	0.705882	1	0.545455	0.6742	0.88
KNN	0.75	0.4	1	0.25	0.426401	0.75
KNN	0.722222	0.375	0.6	0.272727	0.256711	0.545455
KNN	0.75	0.307692	0.666667	0.2	0.261785	0.717308
KNN	0.722222	0.444444	0.8	0.307692	0.366966	0.704013
KNN	0.694444	0.153846	0.5	0.090909	0.102378	0.74
KNN	0.828571	0.571429	0.571429	0.571429	0.464286	0.793367
KNN	0.742857	0.571429	1	0.4	0.525226	0.818333
MLP	0.75	0.4	0.375	0.428571	0.243855	0.748768
MLP	0.888889	0.714286	0.714286	0.714286	0.64532	0.82266
MLP	1	1	1	1	1	1
MLP	0.888889	0.8	1	0.666667	0.755929	0.961806
MLP	0.972222	0.956522	0.916667	1	0.938083	0.989091
MLP	0.944444	0.909091	0.833333	1	0.877058	0.996154
MLP	1	1	1	1	1	1
MLP	0.972222	0.952381	1	0.909091	0.934947	1
MLP	1	1	1	1	1	1
MLP	1	1	1	1	1	1
KNN Average	0.771032	0.429336	0.730476	0.320964	0.371067	0.738936
MLP Average	0.941667	0.873228	0.883929	0.871861	0.839519	0.951848
Naive Bayes Average	0.757381	0.569174	0.584365	0.583227	0.413547	0.774301

There are various measurements to evaluate a model, and here is a basic introduction to them

- ➢ F1: F1 combines precision and recall, with a range between 0 and 1. The closer it is to 1, the better the model performance.
- ➢ Precision: Precision evaluates the accuracy of the model in predicting positive instances. A value closer to 1 indicates better performance in predicting positive instances.
- ➢ Recall: Recall calculates the proportion of correctly identified positive instances among those instances predicted as positive. A value closer to 1 indicates better model performance, suggesting that the model is effective in correctly identifying a higher percentage of positive cases.
- ➢ MCC (Matthews Correlation Coefficient): MCC combines all aspects of model performance, providing a metric for overall performance in the range of -1 to 1. The closer the value is to 1, the better the model performance.
- ➢ AUC: Through the area under the curve, the indicator evaluates the performance of the classification model, with a value closer to 1 indicating better model performance.

We can observe that, across various performance metrics, MLP demonstrates better capabilities. Next, comparing KNN and Naive Bayes Learning, KNN outperforms NBL in precision, but lags behind in other indicators. This suggests that the model tends to be conservative in predicting positive instances. Instances predicted as positive by KNN have a higher chance of being actual positive cases.

Problems

These are the challenges I encountered during the implementation:
1. Initially, this line of code, "X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]," couldn't be executed in any way. Later, it was found that adding ".iloc" was necessary for it to run correctly.
2. Understanding what each function needs to return took quite some time in the beginning, especially in the case of predict_prob. It was only by checking the y_test inside the evaluation that I confirmed this function should return indices and the probability of predicting 1.
3. Implementing binary features for Naive Bayes is much more complex than numerical features. It requires careful pre-planning of variable usage, and the process involves many conversions, making the implementation more challenging than anticipated.
4. MLP requires cumulative training data to make progress. Initially, I placed the initialization of weights inside fit(). However, even with 3000 epochs, the model didn't train effectively. I even discovered that the error had reduced to the order of 10^-8, but the accuracy remained quite low. It was later discovered that each time new data was used to fit the model, the weights were reinitialized, leading to poor training of the model.

Key insights:

1. When building a model, it is crucial to consider various evaluation metrics, especially in the presence of highly imbalanced data. Relying solely on accuracy may not determine the true performance of the model.
2. Preprocessing is a critical step, and blindly imputing values such as 0, median, or mean can introduce issues, especially when dealing with unknown cases. This may lead to suboptimal predictions for instances with missing data.
3. Model parameters vary for different datasets, and experimentation is necessary to determine the optimal settings. However, insights can be gained from parameter tuning. For example, in KNN, a large K may result in poor

predictions, indicating that considering too many neighbors' distances might not accurately represent the class of the instance.

Predictions result for “testWithoutLabel.csv”:

I will choose KNN and MLP for performance measurements.

model	predictions
Naive Bay	[0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0]
KNN	[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
MLP	[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1]