CMPUT 466/566: Project Report Paper

Machine Learning Methods in Cancer Detection

Eden Zhou
University of Alberta
Edmonton, Canada
zhou9@ualberta.ca

INTRODUCTION

In this project, we applied a few supervised learning methods for chest cancer detection. Given some sample data of features related to cancer diagnosis, our trained classifiers should be able to detect whether it is a malignant or benign case.

To have the best performance, a few classifiers based on diverse classification methods of *Naive Bayes (NBs), k-nearest Neighbors (k-NN), Support Vector Machine (SVM), Logistic Regression* and a trivial Baseline Classifier are constructed for analysis and comparison.

For learning optimization, we implement a systematic hyperparameter tuning strategy within a "training/validation-test" framework with the construction of the cross-validation pattern for model learning.

PROBLEM FORMULATION

Data Set

The dataset is taken from *UC Irvine Machine Learning Repository* and named as *Wisconsin Diagnostic Breast Cancer (WDBC)* Data Set.
There are 569 instances in the dataset, and each instance has 30 learnable features: 357 samples within the set have labels of "benign", while the remaining 212 instances have been labelled as "malignant". Each instance in the dataset is identified by a unique ID. More official descriptions of this data set can be found in the file "wdbc.names" under the primary directory.

Input

There is a TSV file under the project package named "wdbc.csv". The main program of this classification task will input and pre-process training and test datasets from this file. And take data segmentation into two parts for "training/validation" and "test" with a ratio of 8:2. Data learned from analysis and predictions will be output to the STDOUT channel of the program with three figures (for each classifier) about relevant classification performances created in the package directory.

Cross-validation

Due to the restriction in data amount, the strategy of *4-fold Cross-validation* is used in model training. The *scikit-learn* built-in method of *model_selection.cross_val_score* is applied and we select k=4 as the hyperparameter in data segmentation: all data in the given training data set are divided into four parts uniformly. For each fold round, one classifier model object will be built, with *(k-1)* parts in all k applied for training and the single part left will be used as the validation data. This strategy will repeat the same process for k times based on different data collections.

Evaluation

The mean of *Balanced Accuracy*, negative *Mean Square Error (MSE),* and the *Mean of F1_micro* score are evaluated in training over rounds of cross-validation and diverse hyperparameters. For MSE and F1_micro, We compute the averaged performance over rounds under the same hyperparameter to draw the figures for hyperparameter-based performance analysis. The comparison of the mean of the validation F1_micro contributes to finding the optimized hyperparameter selection in givens and bringing

specific classifiers with the best performance.

Based on the best hyperparameter choice learned in the previous steps, classifiers with optimized performance are built for making predictions over the classes of the test data set. The overall estimation based on scores of *Recall* and *Precision* will be estimated by the *sklearn* method of *metrics.f1_score*. This is also treated as the measure of success in this project, where the F1 value determines the correctness of classification performance, in other words, the correctness that our parameter-based chest cancer detectors predict.

METHODS

Baseline Classifier

A baseline classifier is created based on simple ideas of mean value comparison (The core idea is taken from *Rocchio Classification*). It simply sums all 30 feature values up and divides them by the feature amount to find the *centroid* for each sample. Then loop over the whole test data set and average all centroid feature values to find the *threshold*. The threshold value is used in classifying: all test instances with centroid values higher than the threshold are considered "malignant", and otherwise labelled as "benign". The F1_score of the baseline classifier is determined to be 0.921 through testing, with a loss of 9 established on *Euclidean Distance*.

Logistic Regression Classifier

Logistic regression is a simple and interpretable algorithm for classification, used to model the relationship between the input features and the probability of the binary outcome. In the case of classification, the logistic regression classifier learns a set of weights for the input features. When new test data is input, these weights are linearly combined with the test data to obtain a z value:

$$z = w_1x_1 + w_2x_2 + \ldots + w_mx_m + b = w^Tx + b$$

where b is the bias term. The z value is then passed through a *sigmoid* function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

which maps the z value to a probability between 0 and 1. This probability represents the likelihood of the input belonging to a certain class.

In the implementation of logistic regression by *sklearn*, *linear_model.LogisticRegression()* takes a parameter C which is the inverse of regularization strength. Smaller C-values lead to stronger regularization by putting more penalties, and vice versa. Regularization is used to prevent *overfitting* issues and improve the generalization performance of the model.

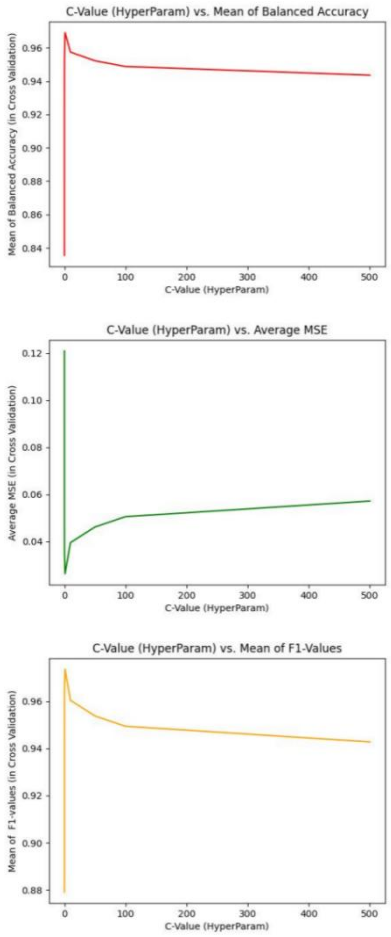

Figure 01: *All three figures illustrate how Logistic Regression model performance changed w.r.t. different hyperparameter elections. From the top to the bottom: Figure 01-1 shows the changes in averaged balanced accuracy w.r.t hyperparameter adjustments; Figure 01-2 shows the changes in training loss w.r.t hyperparameter adjustments; Figure 01-3 shows the changes in avg F1_micro values w.r.t hyperparameter changes.*

The prediction analysis of the classifier found the optimized hyperparameter as C=1, which brings the prediction accuracy of 0.989 in testing,

with the figures above illustrating the changes in model performance (balanced mean accuracy, MSE and mean F1_micro values) with respect to the adjustment of *C*. As value *C* gets larger, the MSE values keep increasing with the minimum present at *C*=1 with about 0.03. Under the same settings, the classifier sees a significant decrease in the Mean F1_micros values and Mean Balanced Accuracies, the trends of reducing decelerate and converge around 0.945 and 0.950, respectively.

k-nearest Neighbors (k-NN) Classifier

The *k-nearest neighbour (k-NN)* algorithm is another widely used classification technique. It operates by comparing the distances between an unknown sample and all other learned samples in the training dataset to determine the class of an unknown sample. The *k* nearest samples with the closest distance to the unknown sample are selected based on a voting rule that follows the majority.

This procedure is executed based on the *neighbors.KNeighborsClassifier()* function in *sklearn*, where the hyperparameter of value *k* specifies the number of neighbors to be considered in *Euclidean distance* measuring. Based on different *k* values, the *k*-NN classifier performs differently in validation accuracy, negative mean squared error, and training loss (shown in *Figure 02* below).

The prediction analysis of the classifier found the optimized hyperparameter as *k*=3, which brings the prediction accuracy of 0.989 in testing, with the figures below displaying the changes in model performance (balanced mean accuracy, MSE, and mean F1_micro values) with respect to the adjustment of *k*. As *k* gets larger, the MSE value initially drops and meets the minimum at *k*=3, but then the error increases steeply. Under the same settings, the classifier sees a considerably steep decreasing trend in the Mean F1_micros values and Mean Balanced Accuracies, with the optimization occurring around 0.960 and 0.955, respectively.

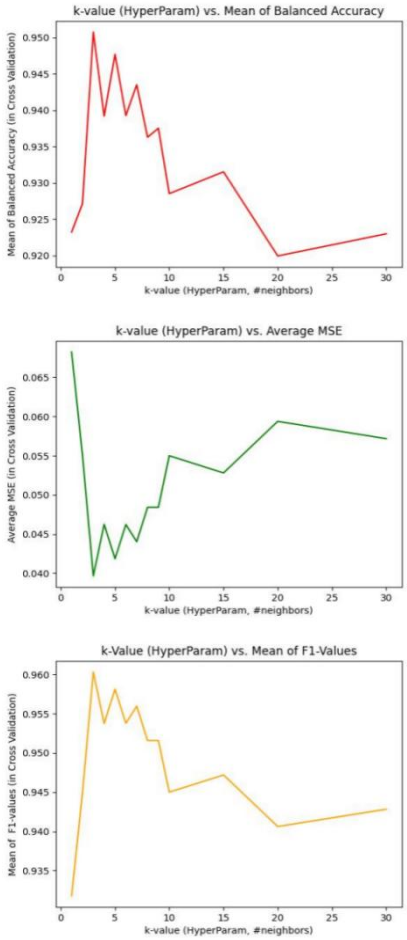

***Figure 02**: All three figures illustrate how k-NN model performance changed w.r.t. different hyperparameter elections. From the top to the bottom: Figure 02-1 shows the changes in averaged balanced accuracy w.r.t hyperparameter adjustments; Figure 02-2 shows the changes in training loss w.r.t hyperparameter adjustments; Figure 02-3 shows the changes in avg F1_micro values w.r.t hyperparameter changes.*

Naive Bayes (NBs) Classifier

Naive Bayes (NBs) is a variety of the *Bayesian* equation that developed upon the simplified version of the Bayesian formula. It is relatively efficient in classification tasks. This method makes the assumption of feature independence, assuming that each feature is independent and uncorrelated, with the equation of:

$$p(y|x) = p(x|y)p(y)$$

Figure 03 below illustrates the NBs classifier performs with respect to different choices of hyperparameters. The cross-validation founds the best hyperparameters of α=100. During the training and validation process, Raising α value results in a decrease of MSE at the initial stage,

while it then increases significantly after touching the minimum point of α=100 with an error value of around 0.0725. Under the same settings, the Mean F1_micros values and Mean Balanced Accuracies see slight growth at the beginning until reaching the optimal at α=100 with mean F1_micros of 0.9275 and Mean Balanced Accuracies of 0.923. However, the line keeps dropping after the peak till the end.

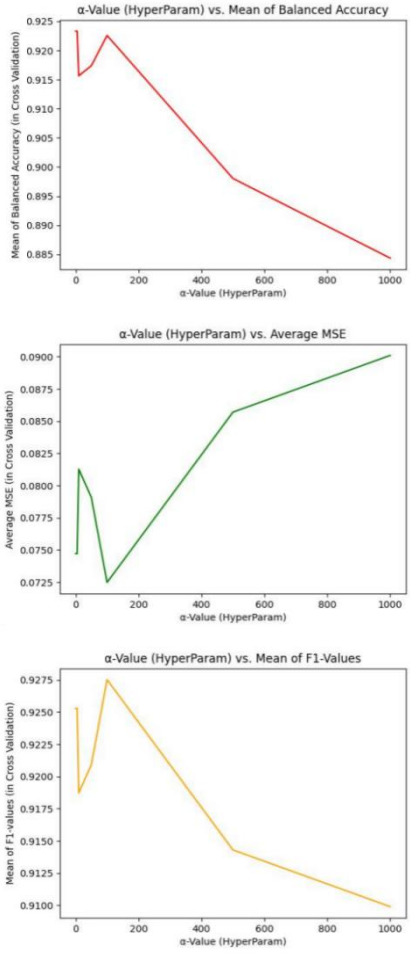

Figure 03: *All three figures illustrate how NBs model performance changed w.r.t. different hyperparameter elections. From the top to the bottom: Figure 03-1 shows the changes in averaged balanced accuracy w.r.t hyperparameter adjustments; Figure 03-2 shows the changes in training loss w.r.t hyperparameter adjustments; Figure 03-3 shows the changes in avg F1_micro values w.r.t hyperparameter changes.*

Support Vector Machine (SVM) Classifier

Support Vector Machine (SVM) was considered as one of the most powerful classification models in basic ML approaches that maps samples' features to points in a vector space. It learned to construct a separating line of *decision boundary* that contributes to differentiating points into distinct categories. This line can be easily applied to new feature points, which caused the SVM classifier operates efficiently in high dimensional conditions and be effective in non-linear classification tasks.

To implement an SVM classifier, *svm.SVC()* function of *sklearn* is utilized. The function has a regularization term of *C* which serves as a hyperparameter to control the intensity. Similar to what we explained for Logistic Regression, an advancing *C* value leads to more powerless regularization and vice versa. *Figure 04* below explains how various selections of the hyperparameter determine the performance of the SVM classifier.

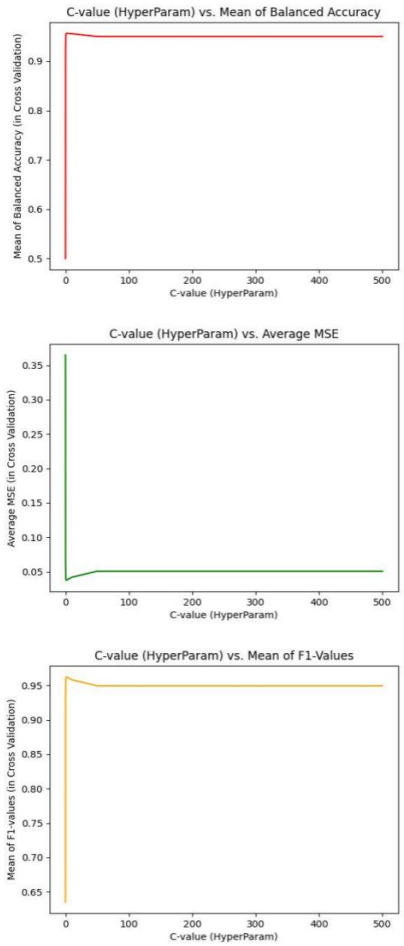

Figure 04: *All three figures illustrate how SVM model performance changed w.r.t. different hyperparameter elections. From the top to the bottom: Figure 04-1 shows the changes in averaged balanced accuracy w.r.t hyperparameter adjustments; Figure 04-2 shows the changes in training loss w.r.t hyperparameter adjustments; Figure 04-3 shows the changes in avg F1_micro values w.r.t hyperparameter changes.*

C=1 is accepted as the optimized choice for the regularization term in the validation process. Enhancing the *C* value results in a slight

decrease in the mean balanced accuracy and the mean F1_micros values, then both converge to around 0.97 and 0.95, respectively. The same experiment sees training MSE continues boosting and converges to around upper 0.05. The minimal MSE value during the upward trend occurs at C=1 with about 0.03.

RESULTS

According to the test results learned from diverse approaches of classifiers, the *Logistic Regression* Classifier, the *k-nearest Neighbors (k-NN)* Classifier, and the *Support Vector Machine (SVM)* Classifier perform similarly in terms of final F1-Score in tests, with values ranging from around 0.9890 to 0.9892. By contrast, the *Naive Bayes* classifier presents poorly with a test F1-Score of only about 0.9474. However, all four classification methods of Machine Learning deliver much more powerful predictions in classification than the *Baselines* classifier (based on *Rocchio's* Ideas).

```
* Baselines - The overall prediction analysis is:
*     finalTestFScore --> 0.9210526315789473
*     finalTestLoss --> 9
-----------------------------------------------------------------------------------------
* LogRegModel - The overall prediction analysis is:
*     crossValidationSettings --> 4-folds, with estimations of ('neg_mean_squared_error', 'f1_micro', 'balanced_accuracy')
*     bestHyperParamFound --> C=1
*     finalTestFScore --> 0.989247311827957
-----------------------------------------------------------------------------------------
* KNNModel - The overall prediction analysis is:
*     crossValidationSettings --> 4-folds, with estimations of ('neg_mean_squared_error', 'f1_micro', 'balanced_accuracy')
*     bestHyperParamFound --> k=3 (neighbors)
*     finalTestFScore --> 0.989010989010989
-----------------------------------------------------------------------------------------
* NBModel - The overall prediction analysis is:
*     crossValidationSettings --> 4-folds, with estimations of ('neg_mean_squared_error', 'f1_micro', 'balanced_accuracy')
*     bestHyperParamFound -->  α=100
*     finalTestFScore --> 0.9473684210526316
-----------------------------------------------------------------------------------------
* SVMModel - The overall prediction analysis is:
*     crossValidationSettings --> 4-folds, with estimations of ('neg_mean_squared_error', 'f1_micro', 'balanced_accuracy')
*     bestHyperParamFound --> C=1
*     finalTestFScore--> 0.989247311827957
```

Figure 05: *The overall test performance according to different classifiers (Baseline, Logistic Regression, k-NN, NBs, and SVM). Except for the Baseline classifier, the measuring strategies, the optimized hyperparameters, and the final test F1-Scores of each classifier are listed. The prediction F1-Score and Euclidean Distance-based error are also computed for the baselines.*

As for hyperparameter selections, the optimal choices vary for each classifier: Baselines: F1-Score of 0.9211 and loss of 9 (p.s., no hyperparameter used for the baseline setting); Logistic Regression Model is optimized with C=1; k-NN model works the best with k=3; NBs Model holds α=100 for learning optimization; and C=1 helps optimize the SVM Classifier. More details can be found in *Figure 05*, which is the overall output of solving this classification problem.

REFERENCES

[1] UC Irvine Machine Learning Repository *Breast Cancer Wisconsin* (Diagnostic) Data Set. 1995

[2] scikit-learn *neighbors.KNeighborsClassifier* Official Documentation

[3] scikit-learn *naive_bayes.CategoricalNB* Official Documentation

[4] scikit-learn *svm.SVC* Official Documentation