Lin Ouyang

CS 5350/6350 Machine Learning,

Fall 2024 – Projects

Final Project Report

**1. Problem definition and motivation**

In this project, I selected the competitive project which is a Kaggle competition. The objective of the competition is to predict whether a resident’s annual income exceeds $50,000 based on various demographic and employment-related attributes. This binary classification task involves handling a dataset with a mixture of numerical and categorical features, making it a practical scenario for applying and evaluating machine learning techniques.

I chose this competition as it aligns closely with the topics covered in my course. By participating, I aim to deepen my understanding of machine learning concepts and their practical implementations. The competition not only provides a challenging environment to test theoretical knowledge but also offers an opportunity to explore advanced techniques and model optimization strategies

To address the problem, I employed four machine-learning methods learned from the course:

1. Decision Tree: A foundational model for classification tasks that splits data based on feature importance.
2. Random Forest: An ensemble learning method that combines multiple decision trees to improve prediction accuracy and reduce overfitting.
3. Batch Gradient Descent: A numerical optimization technique to minimize the cost function, particularly useful for linear models.
4. Dual Support Vector Machine: A classification approach that finds the optimal hyperplane in a transformed feature space to separate data points from different classes.

Through this project, I aim to evaluate the performance of these methods on a real-world dataset and enhance my ability to choose, implement, and tune models effectively for machine learning challenges.

**2.My solutions**

**Data Preprocessing**

One key aspect of my approach was effective data preprocessing, particularly handling numerical feature transformations and missing values:

1. **Binarization of Numerical Features**: For numerical features, I transformed them into binary values based on the median value in the training dataset. Specifically, values greater than the median were converted to 1, while those less than or equal to the median were converted to 0.
2. **Handling Missing Values**: Initially, I treated the missing value symbol ("?") as a valid categorical value, making it part of the attribute's possible values during training and testing. However, after analyzing the decision tree model's performance and the data, I revised this strategy in later submissions by replacing missing values with the mode (most frequent value) of the corresponding attribute in the training dataset.

**Decision Tree Model**

In the decision tree model, I adopted a binary classification approach:

1. Each test sample's target variable was directly predicted as either 1 or 0.
2. Splits in the decision tree were based on metrics such as information gain, Gini index, or majority error. I also set a maximum depth to prevent overfitting​​.

**Random Forest Model**

For the random forest model, I explored a probabilistic prediction approach:

1. Each tree predicted the probability of the target variable being 1. These probabilities were aggregated across all trees to produce a final probability value, ranging from [0, 1].
2. This method enhanced the model's generalization ability while reducing the variance of individual decision trees​.

I implemented the following key strategies:

1. **Bootstrap Sampling**: To improve the diversity of the trees, each tree in the forest was trained on a bootstrapped sample of the training dataset. This involves sampling with replacement, ensuring that each tree is trained on a unique subset of the data.
2. **Feature Subset Selection**: For each split within a tree, a random subset of features was selected. This technique reduces the correlation between trees, further enhancing the ensemble’s generalization capabilities​.
3. **Tree Construction**: Each tree in the forest was built using a decision tree algorithm based on the ID3 method. The split criterion was entropy, and no maximum depth was imposed, allowing trees to grow fully based on the available data​.
4. **Prediction Aggregation**: During prediction, each tree produced a probability for the target variable being 1. These probabilities were averaged across all trees to generate a final prediction for each sample. This probabilistic approach improved the model's robustness and reduced variance caused by individual trees​.

**Batch Gradient Descent**

I implemented a batch gradient descent model to optimize the parameters of logistic regression:

1. **Loss Function**: I used a cross-entropy loss with L2 regularization to prevent overfitting.
2. **Standardization**: Input features were standardized to have a mean of 0 and a standard deviation of 1.
3. **Learning Rate**: A fixed learning rate was used, and the model converged to stable weight vectors after multiple iterations​.

**Dual SVM**

For the support vector machine, I used a dual-form optimization algorithm:

1. A kernel function (Gaussian kernel) was employed to capture nonlinear relationships in the data.
2. The dual problem was solved using Lagrange multipliers with constraints to optimize the model.

I introduced a penalty parameter C to balance maximizing the margin and penalizing misclassified points. During the prediction phase, decision function values were converted to probabilities, representing the likelihood of the target variable being 1​.

**3.Experimental results**

**Decision Tree Model: Submissions 1 to 3**

In my initial attempts, I used a basic decision tree model with a maximum depth of 14. For the first submission, I used the Gini index as the splitting criterion, while the second submission used entropy, offering different metrics to evaluate node purity.

1. First submission: Using the Gini index, the model achieved a score of 0.71983 on the Kaggle leaderboard.
2. Second submission: Switching to entropy slightly improved the score to 0.72315, indicating that entropy was more effective in this case.

In the third submission, I further optimized the decision tree by improving the handling of missing values. Instead of treating "?" as a separate attribute value, I filled the missing entries with the mode (most frequent value) of the respective attribute.

1. Third submission: This adjustment resulted in a slight improvement, achieving a score of 0.72335.

**Random Forest Model: Submissions 4 to 13**

For the random forest model, I trained 500 trees for each forest without setting a maximum depth to capture complex patterns in the data. The feature subset size was set to 6, and I used bootstrapping to draw samples with replacement.

1. Fourth submission: Using 10,000 training samples, the model achieved a significant improvement with a score of 0.86783.
2. Fifth submission: Increasing the training sample size to 20,000 resulted in a slight drop in performance, with a score of 0.85245.

To investigate the effect of training sample size on model performance, I adjusted the sample size in subsequent submissions:

1. Sixth submission: Using 15,000 training samples, the model scored 0.85806.
2. Seventh submission: Reducing the sample size to 5,000 improved the score to 0.87771.
3. Eighth submission: Further reducing the sample size to 1,000 achieved the best score so far, 0.88871.
4. Ninth submission: Reducing the sample size to 500 resulted in a score of 0.88869, showing a marginal decrease.

Additionally, I experimented with the feature subset size and the number of trees:

1. Tenth submission: Reducing the feature subset size to 4, with 1,000 samples and 500 trees, achieved a score of 0.89019.
2. Eleventh submission: Training 1,000 trees with 1,000 samples and a feature subset size of 4 resulted in a score of 0.89032.
3. Twelfth submission: Using 300 trees, 1,000 samples, and a feature subset size of 4, the score improved to 0.89070.
4. Thirteenth submission: Increasing the feature subset size to 8 with 300 trees and 1,000 samples slightly reduced the score to 0.88766.

**Batch Gradient Descent Model: Submissions 14 and 15**

I implemented logistic regression with batch gradient descent, using the following configurations:

* Learning rate: 0.01
* L2 regularization strength: 0.01

The results were as follows:

1. Fourteenth submission: Using 2,000 epochs, the model achieved a score of 0.83795.
2. Fifteenth submission: Reducing the number of epochs to 1,000 resulted in a score of 0.83725, showing stable but slightly lower performance with fewer iterations.

**Dual SVM Model: Submissions 16 to 20**

Due to the long training time for large datasets, I trained the Dual SVM model on smaller subsets of data.

1. Sixteenth submission: Using 500 training samples with binary outputs (0 or 1), the model scored 0.68493.
2. Seventeenth submission: Increasing the training sample size to 1,000 achieved a score of 0.68402.

I then adjusted the model to output probabilities instead of binary predictions:

1. Eighteenth submission: Using 500 training samples, the score improved to 0.82408.
2. Nineteenth submission: With 1,000 training samples, the model achieved a score of 0.83328.
3. Twentieth submission: Increasing the sample size to 3,000 resulted in a score of 0.8330.

The experimental results demonstrate the effectiveness of applying diverse machine-learning models and parameter tuning strategies to a real-world dataset. Among the models explored, the random forest model consistently outperformed other methods, with its probabilistic prediction approach and feature subset selection contributing significantly to its success. The highest score of 0.89070 was achieved using 300 trees, 1,000 samples, and a feature subset size of 4.

The batch gradient descent model and the Dual SVM model, while not achieving the highest scores, provided valuable insights into the trade-offs between training time and performance, particularly for logistic regression and kernel-based methods. Notably, converting the Dual SVM's outputs to probabilities resulted in a substantial improvement in its score.

These results highlight the importance of careful preprocessing, model selection, and hyperparameter tuning in achieving competitive performance on complex machine-learning tasks. Furthermore, the experiments underscore the necessity of balancing model complexity and training efficiency, particularly when dealing with large datasets.

**4.If I have much more time**

If I had more time, I would try additional types of models, including some that I learned in class but did not have time to implement, as well as advanced models commonly used in real-world applications.

In addition, I am very interested in exploring other methods for data preprocessing. Specifically, I would like to experiment with more advanced ways of handling numerical features, beyond the current approach of binarizing them based on the median.

**Github**

https://github.com/ouyangl3/CS-5350-6350-Machine-Learning-Fall-2024---Projects