Data Mining CS-535

Assignment 2

Sarthak Zende B01035919

Questions:

This assignment focuses on classification study. There are three files associated with this assignment: adult.data, adult.test, and adult.names. These data are obtained from the public UCI data archive. There are 15 columns in the data set. The first 14 columns are attributes, and the last column is the class label.

1. (10 pts.) Clean up the data set. There are tuples in the data set in which there are missing values. Based on what you have learned in the course, propose and implement a method to clean up the data set. Note that the data set includes adult.data and adult.test both files.

- **Step 1: Clean up the dataset** (Source code attached in source.ipynb file)

First, we need to identify and handle missing values in both the adult.data and adult.test files. Based on what I learned from the course for data cleaning:

1. Ignoring the Tuple.
2. Fill in missing values manually.
3. Imputing missing values automatically with the mean, median, or mode (for numerical data) or the most frequent value (for categorical data).

Given the nature of this dataset, it's common to find missing values denoted by a specific marker, like "?". From the 3 strategies discussed in class we will go with the most suitable one which is Imputing missing values automatically.
There are missing values in workclass, occupation, and native_country in both the training and test datasets. For simplicity and to retain as much data as possible, we'll impute these missing values. Since workclass and occupation are categorical, we'll fill their missing values with the mode (most frequent value) of their respective columns. For native_country, also categorical, we'll apply the same strategy. This approach keeps the data distribution similar to the original dataset without discarding valuable data.

Let's proceed with the data cleaning process: (Source code attached in source.ipynb file)

Impute missing values for workclass, occupation, and native_country with their modes. The missing values in both the training and test datasets have been successfully imputed. There are no missing values in the datasets now.

Before	**After**
(age 0 workclass 1836 fnlwgt 0 education 0 education_num 0 marital_status 0 occupation 1843 relationship 0 race 0 sex 0 capital_gain 0 capital_loss 0 hours_per_week 0 native_country 583 income 0 dtype: int64, age 0 workclass 963 fnlwgt 0 education 0 education_num 0 marital_status 0 occupation 966 relationship 0 race 0 sex 0 capital_gain 0 capital_loss 0 hours_per_week 0 native_country 274 income 0 dtype: int64)	(age 0 workclass 0 fnlwgt 0 education 0 education_num 0 marital_status 0 occupation 0 relationship 0 race 0 sex 0 capital_gain 0 capital_loss 0 hours_per_week 0 native_country 0 income 0 dtype: int64, age 0 workclass 0 fnlwgt 0 education 0 education_num 0 marital_status 0 occupation 0 relationship 0 race 0 sex 0 capital_gain 0 capital_loss 0 hours_per_week 0 native_country 0 income 0 dtype: int64)

2. (30 pts.) Implement a classification method either you have learned from this course or from the literature. Use adult.data as the training dataset and use adult.test as the evaluation dataset to report the classification error rate. The classification error rate is defined as the number of the truth incorrect labels reported by your method in the evaluation dataset divided by the number of the truth labels in the evaluation dataset (i.e., the total number of tuples after data cleaning in the evaluation dataset). Turn in the source code of your implementation with documentation.

- **Step 2: Implement a Classification Method**

In the course we have learned about many classification methods like

1. SVMs
2. Neural Networks
3. Lazy Learners (k-NN)
4. Bayesian Classifiers
5. Decision Trees
6. Random Forest

But I have decided that for our this datasets Random Forest is the best choice since it has robustness to overfitting, Handling Mixed Data Types, Feature Importance ,Minimal Preprocessing unlike the above algorithms that require extensive preprocessing (e.g., scaling for SVMs), Random Forest requires less preprocessing, simplifying the pipeline. Ensemble Learning- As an ensemble method, Random Forest combines the predictions from multiple decision trees to improve prediction accuracy and control overfitting.

Implementation of Random Forest:

1. Preprocess the data
2. Split the data
3. Train the model
4. Evaluate the model

Results : (Source Code attached in source.ipynb file.)

Classification error rate (as a decimal):	0.14882378232295312
Classification error rate (percentage):	**14.88%**
Number of correct predictions:	13858
Number of incorrect predictions:	2423
Total number of instances in the test dataset:	16281

3. (10 pts.) Compare your classification error rate with the reported error rates in adult.names for the most well-known classification methods, and analyze the differences. (10 pts.) Suggest any improvements if there is a difference.

- **Step 3.1: Compare Classification error rate and Analyze the differences**

Algorithm	Reported Error Rate	My Model Error Rate	Remarks and Potential for Improvement
C4.5	15.54%	14.88%	Slightly better. Could further tune hyperparameters.
C4.5-auto	14.46%	14.88%	Slightly worse. Consider feature engineering.
C4.5 rules	14.94%	14.88%	Comparable. Ensemble methods like AdaBoost could be explored.
NBTree	14.10%	14.88%	Worse. Hybrid models or advanced ensembles may improve results.
FSS Naive Bayes	14.05%	14.88%	Worse. Investigate feature independence assumptions.
IDTM (Decision Table)	14.46%	14.88%	Slightly worse. Dimensionality reduction could help.
Random Forest	14.88%	14.88%	Use cross-validation to ensure robustness against overfitting.

Difference:

1. Comparison with C4.5 and Variants: My error rate is slightly lower than the C4.5 decision tree but not as low as the C4.5-auto and IDTM. This could be due to the Random Forest classifier's ensemble nature, which generally improves upon the decision tree's performance through averaging and reducing variance.

2. Comparison with Naive Bayes Variants: FSS Naive Bayes seems to perform slightly better than my Random Forest model. This could be because Naive Bayes performs well with independent features and might be more robust to the specific distributions in the dataset.

3. Comparison with NBTree: My model has a slightly higher error rate than NBTree, which is a hybrid model that combines Naive Bayes and decision trees. This

indicates that combining the probabilistic approach of Naive Bayes with the hierarchical structure of decision trees might capture the patterns in the data more effectively.

❖ **Step 3.2: Proposed Improvements** (Source code attached in source.ipynb file)

There are several ways we could do this as mentioned in course slides and literature:

1. Hyperparameter Tuning
2. Feature Engineering
3. Algorithm Tuning
4. Ensemble Learning
5. Data Cleaning
6. Cross-Validation
7. Dimensionality Reduction

Based on the performance of my Random Forest classifier and considering the balance between potential impact and implementation complexity, let's go with Hyperparameter tuning :

1. **Hyperparameter Tuning:** This involves optimizing the model's settings to improve its performance. For Random Forest, important settings like the number of trees (n_estimators), tree depth (max_depth), and features considered per split (max_features) can be adjusted. Techniques such as grid search and random search, alongside cross-validation, help identify the best hyperparameter values systematically.

❖ **Step 3.2: Hyperparameter Tuning** (Source code in source.ipynb)

For hyperparameter tuning, we'll use GridSearchCV from sklearn.model_selection. This allows us to search across a range of hyperparameter values and find the best combination based on cross-validated performance. We'll focus on a few key hyperparameters for simplicity:

- n_estimators: Number of trees in the forest.
- max_depth: Maximum depth of the tree.
- max_features: Number of features to consider when looking for the best split.

Results after performing Hyperparameter Tuning :
Best parameters: {'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 200}
New classification error rate (as a decimal): 0.13635526073337023
Classification Error Rate : 13.64%

As you can see after we implemented our proposed method of hyperparameter tuning on the model we got better results than all of the other reported model results.
The results show that the best parameters for your Random Forest model are max_depth of 20, max_features of 'sqrt', and n_estimators of 200. Using these parameters, I achieved a **new classification error rate of approximately 13.64%**, which is an improvement over my **previous error rate of approximately 14.88%**.

Hence we have improved the performance of our Random Forest model by adjusting its hyperparameters, and as a result, it outperformed previous models that have been published, with an error rate of 13.64%.

4. (20 pts.) Randomly sample the training data set adult.data X% to obtain a new, down-sampled training data set; then use the down-sampled training dataset to train the classifier from (2) above and use the same evaluation data set adult.test to record the error rate. Repeat the whole process (i.e., randomly sample training data set --- train the classifier --- compute the error rate) five times and report the mean and the deviation for the error rate for each X when X is taken from 50, 60, 70, 80, and 90. Give an analysis on the reported results. Please note that when you down sample a training data set, you need to make sure that you randomly down sample data samples from each class with the given parameter of the sampling rate, i.e., X%.

- **Step 4: Randomly downsampling and Re-training the dataset**

(Source code attached in source.ipynb file)

Results:

Sampling Rate: 50% - Mean Error Rate: 0.1390 [13.9%], Standard Deviation: 0.0014
Sampling Rate: 60% - Mean Error Rate: 0.1381, [13.81%] Standard Deviation: 0.0004
Sampling Rate: 70% - Mean Error Rate: 0.1378, [13.78%] Standard Deviation: 0.0004
Sampling Rate: 80% - Mean Error Rate: 0.1371, [13.71%] Standard Deviation: 0.0006
Sampling Rate: 90% - Mean Error Rate: 0.1365, [13.65%] Standard Deviation: 0.0004

Analysis:

1. Mean Error Rates Analysis:
    a. Decreasing Error Rates: As the sample rate rises from 50% to 90%, the mean error rate continuously falls from 13.90% to 13.65%. This trend suggests that having more data available for training enhances the model's accuracy. The increased dataset size provides more information for the model to learn from, resulting in more accurate predictions.
    b. Impact of Data Volume: The steady improvement in error rates as data volume increases supports the idea that larger training sets might lead to improved model performance. This is especially important for models like Random Forest, which use diversity in the training samples to create a more generic ensemble.

2. Standard Deviation Analysis:
    a. Low Variability: The standard deviations range from 0.0004 to 0.0014, indicating low variability in the model's performance across different iterations of down-sampling for the same rate. This suggests that the

model's performance is relatively stable, not highly sensitive to the specific samples chosen in each down-sampling iteration.

b. Consistency Across Sampling Rates: The consistency in low standard deviations as the sampling rate increases further indicates that the model's performance is robust across different training set sizes. Even as the amount of data changes, the performance of the model does not fluctuate wildly, which is desirable for reliable predictions.

Conclusion:
This step underscores the importance of training set size in achieving optimal model performance, with a clear trend showing improved accuracy as more data is used for training. The low variability in performance across iterations at each sampling rate also highlights the Random Forest model's robustness, suggesting that it can maintain consistent performance even with variations in training data volume.

However, the improvements in error rates are relatively modest and the already low standard deviations also suggest diminishing returns as the dataset size increases beyond a certain point. While larger datasets do contribute to improved model accuracy, the extent of improvement decreases, implying a balance must be achieved between acquiring more data and the computational cost of processing large datasets.

5. (20 pts.) It is observed that all the classic classifiers are far from being perfect. Propose a solution that uses one classifier and the given training data set adult.data, if we use the same evaluation data set adult.test, we can beat all the classic classifiers reported in adult.names.

- **Step 5: Proposed Single Classifier that outperform other classic classifiers**

(Source code attached in source.ipynb file)

To surpass the performance of classic classifiers and achieve a lower error rate with a single classifier on the `adult.data` and `adult.test` datasets, I executed the following streamlined steps. Previously I achieved this with Random forest and performing Hypertuning but since it is ensemble method we cannot use it for Q.5

Proposed Solution -

1. **Data Preprocessing:**

- Encoded categorical features using one-hot encoding.
- Applied standardization to continuous features as part of preprocessing strategy.

2. **Pipeline Creation:**

- Developed a `Pipeline` integrating preprocessing and the Decision Tree Classifier to ensure consistent data treatment and avoid data leakage.

3. **Model Selection:**

- Selected the Decision Tree Classifier after trial and testing with other models like SVM , Logistic Regression , Neural Networks.

4. **Hyperparameter Tuning:**

- Utilized `RandomizedSearchCV` to efficiently explore and optimize key hyperparameters: `max_depth`, `min_samples_split`, and `min_samples_leaf`.

5. **Performance Evaluation:**

- Achieved an optimized error rate of 0.1376 (13.76%), demonstrating superior performance over the classic classifiers reported in `adult.names`.

Results:

Best parameters found: {'classifier__max_depth': 10, 'classifier__min_samples_leaf': 16, 'classifier__min_samples_split': 14}
Best cross-validation accuracy: 0.8571298293603682
Optimized Decision Tree Test Accuracy: 0.8624
Optimized Decision Tree Test Error Rate: 0.1376 [13.76%]

Conclusion:
Hence we have achieved better performance **(13.76%) with Decision tree** than all the classic classifiers mentioned in adult.names . My Decision tree before hyperparameter tuning was giving an 18% error rate which I reduced to 13.76% by performing a series of steps like Hyperparameter tuning.

References :

1. Course Slide Jan 25 Data Cleaning and Preprocessing
2. Course Slide Feb 13 Classification
3. Course Slide Feb 20 SVM
4. Course Slide Mar 12 Model Selection
5. Random Sampling- https://www.youtube.com/watch?v=MfkJu7J1LE4
6. Hypertuning - https://towardsdatascience.com/bayesian-optimization-for-hyperparameter-tuning-how-and-why-655b0ee0b399
7. Random Forest - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
8. Decision Tree - https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial