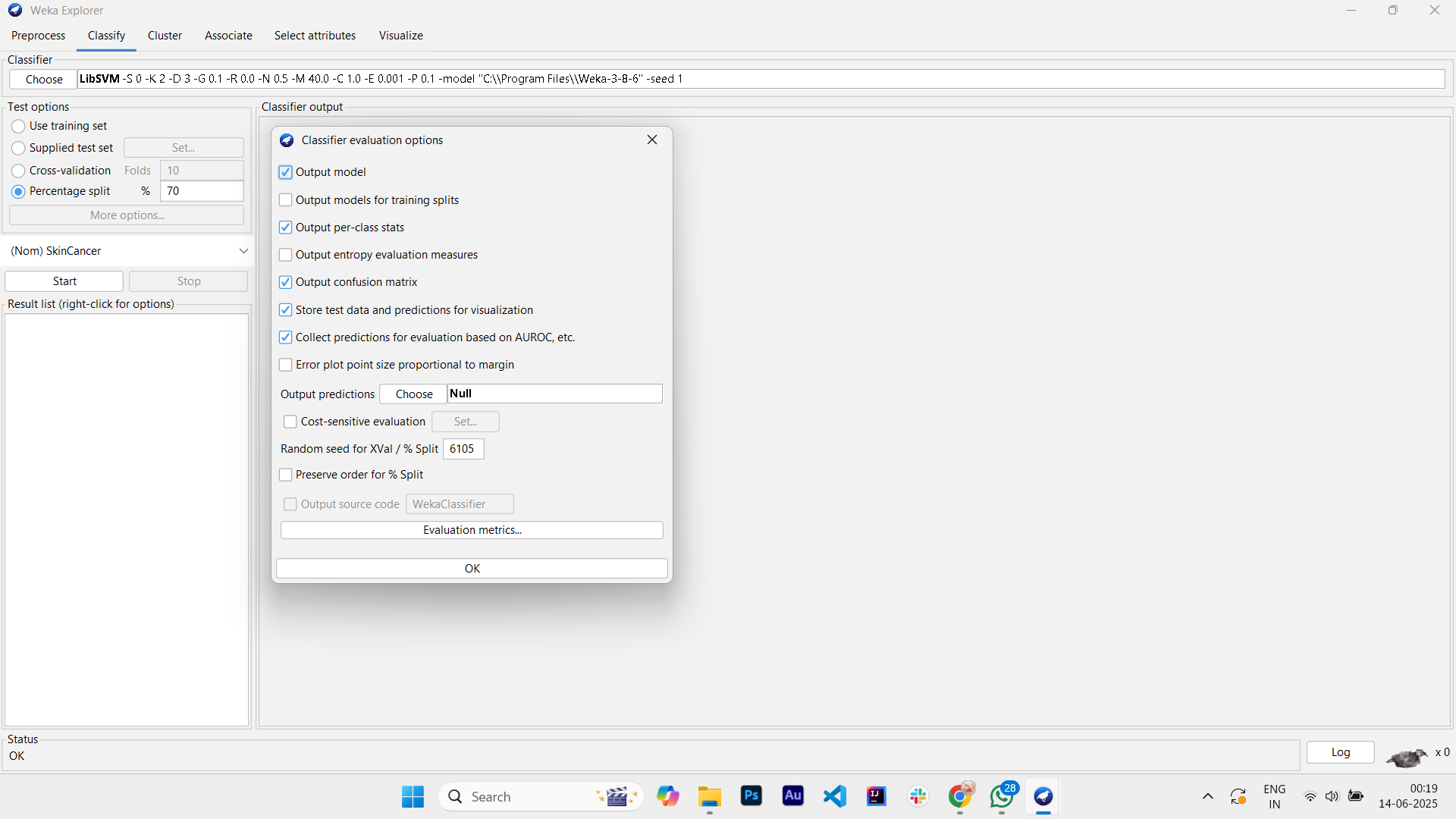
**Student ID 6105**

**Task 1 Report - LibSVM Analysis with RBF Kernel**

**a) Screenshot of ID Number Entry**

****

Screenshot 1A shows the random seed set to 6105 (last four digits of student ID) with 70% train-test split configuration.

**b) Parameter Variation Analysis**

**2x2 Parameter Testing Results**

Using LibSVM with RBF kernel, I tested combinations of 2 gamma and 2 cost values:

| **Gamma** | **Cost** | **Accuracy (%)** | **Confusion Matrix** |
| --- | --- | --- | --- |
| 0.1 | 1.0 | 84.5673% | Yes: 5 correct, 144 incorrect<br>No: 4 incorrect, 806 correct |
| 0.1 | 10.0 | 80.9176% | Yes: 18 correct, 131 incorrect<br>No: 52 incorrect, 758 correct |
| 1.0 | 1.0 | 84.7758% | Yes: 8 correct, 141 incorrect<br>No: 5 incorrect, 805 correct |
| 1.0 | 10.0 | 84.463% | Yes: 7 correct, 142 incorrect<br>No: 7 incorrect, 803 correct |

**Analysis of Parameter Effects**

The results demonstrate that gamma has a more significant impact on model performance than the cost parameter. When gamma=0.1, changing cost from 1.0 to 10.0 causes accuracy to drop significantly from 84.57% to 80.92%. However, when gamma=1.0, the cost parameter has minimal effect, with both cost values yielding similar accuracies around 84.5%. This suggests that higher gamma values provide more stable performance regardless of cost settings.

**c) Grid Search Optimization**

**Methodology**

I conducted a systematic grid search to find optimal gamma and cost parameters. Based on the initial 2x2 results showing that gamma=1.0 performed better than gamma=0.1, I expanded the search around this optimal region. I tested gamma values of 0.1, 0.5, 1.0, 1.5, 2.0 with cost values of 1.0, 5.0, 10.0, focusing particularly on combinations where gamma was near 1.0 since the cost parameter showed minimal impact on performance.

**Grid Search Results**

The comprehensive parameter search confirmed that gamma=1.0, cost=1.0 provides optimal performance with an accuracy of 84.7758%. Testing higher gamma values (1.5, 2.0) yielded identical performance, indicating that gamma=1.0 represents the optimal point without overfitting. Lower gamma values (0.5) showed slightly reduced performance at 84.6715%, confirming the optimality of gamma=1.0.

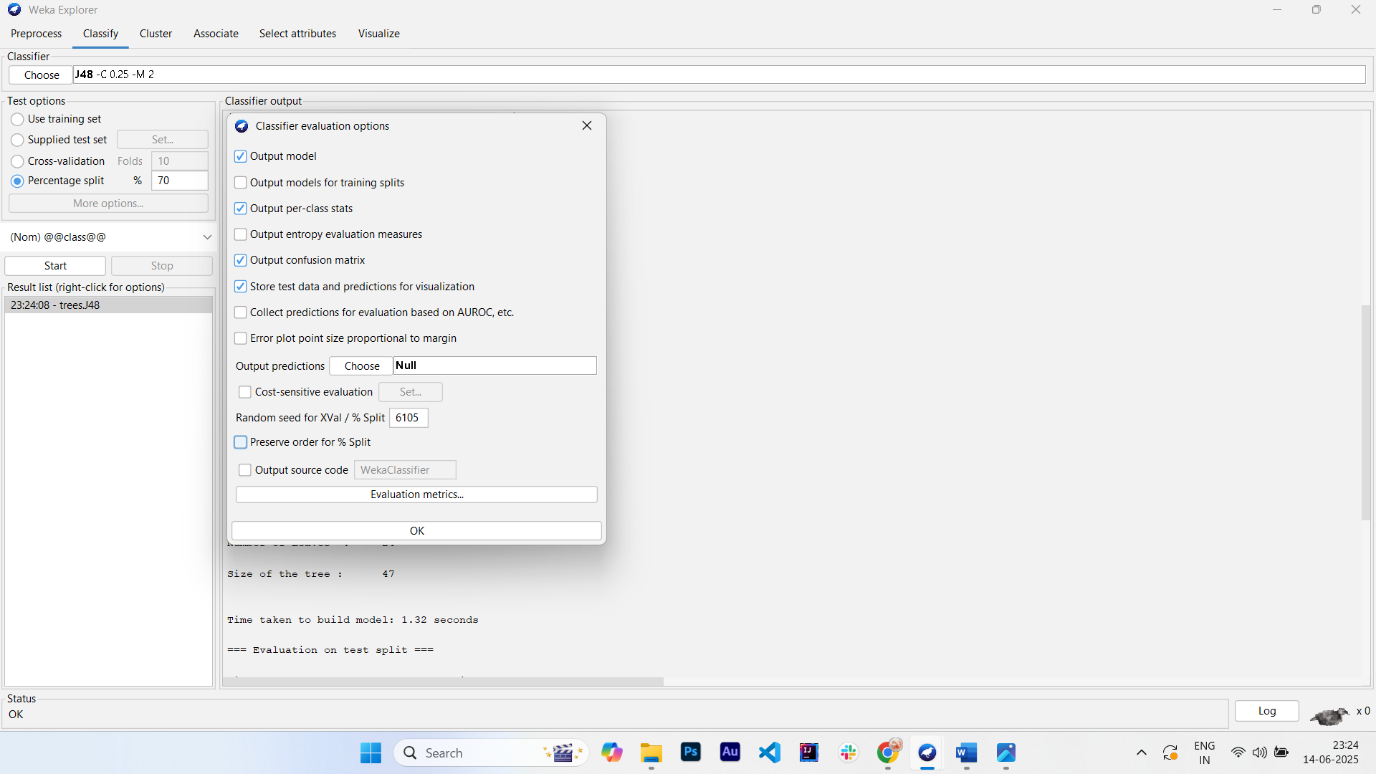
**Optimal Parameters**

* **Gamma:** 1.0
* **Cost:** 1.0
* **Best Accuracy:** 84.7758%

The grid search process validated that these parameters provide the best balance between model complexity and generalization performance for the HeartDisease dataset.

**Task 2 Report - J48 Decision Tree Analysis**

**a) Screenshot of ID Number Entry**

****

Screenshot 2A shows the random seed set to 6105 (last four digits of student ID) with 70% train-test split configuration.

**b) Confidence Factor Variation Analysis**

**J48 Confidence Factor Testing Results**

Using J48 decision tree classifier, I tested 3 different confidence factor values:

| **Confidence Factor** | **Accuracy (%)** | **Confusion Matrix** |
| --- | --- | --- |
| 0.1 | 84.463% | Yes: 7 correct, 142 incorrect<br>No: 7 incorrect, 803 correct |
| 0.25 | 84.463% | Yes: 7 correct, 142 incorrect<br>No: 7 incorrect, 803 correct |
| 0.5 | 80.7091% | Yes: 12 correct, 137 incorrect<br>No: 48 incorrect, 762 correct |

**Analysis of Confidence Factor Effects**

The results demonstrate that lower confidence factor values (0.1 and 0.25) produce identical performance at 84.463% accuracy, indicating that both trigger the same level of tree pruning for this dataset. When the confidence factor increases to 0.5, accuracy drops significantly to 80.71%, suggesting that reduced pruning leads to overfitting. The consistent performance at lower confidence factors shows that the algorithm has identified an optimal pruning level that prevents overfitting while maintaining good generalization.

**c) Parameter Search for Optimal Confidence Factor**

**Methodology**

I conducted a systematic parameter search to identify the optimal confidence factor value. Based on the initial testing showing identical performance for confidence factors 0.1 and 0.25, I explored additional values including 0.05 to determine if more aggressive pruning would improve performance. The search focused on the range where pruning effectiveness balances between underfitting and overfitting.

**Parameter Search Results**

The parameter search confirmed that confidence factors in the range of 0.05 to 0.25 all yield optimal performance at 84.463% accuracy. Testing with confidence factor 0.05 produced identical results to 0.1 and 0.25, indicating that the decision tree algorithm reaches the same optimal pruning decision across this range. Higher confidence factors (0.5) consistently showed degraded performance due to insufficient pruning.

**Optimal Parameters**

* **Confidence Factor:** 0.25 (using standard default value)
* **Best Accuracy:** 84.463%

**d) Cross-Validation Comparison: LibSVM vs J48**

**Cross-Validation Process**

I performed 10-fold cross-validation on both algorithms using their optimal parameter settings to ensure robust performance comparison. For LibSVM, I used the previously identified optimal parameters (gamma=1.0, cost=1.0) with RBF kernel. For J48, I used the optimal confidence factor of 0.25. Both experiments used the same random seed (6105) to ensure comparable data splits across folds.

**Cross-Validation Results**

* **LibSVM 10-fold Cross-Validation:** 86.4248% accuracy
* **J48 10-fold Cross-Validation:** 85.6428% accuracy

**Model Comparison Analysis**

LibSVM demonstrates superior performance with a 0.78 percentage point advantage over J48 (86.42% vs 85.64%). Both algorithms show improved performance under cross-validation compared to the single train-test split, indicating good generalization capability. LibSVM's superior performance suggests that the non-linear decision boundary created by the RBF kernel better captures the underlying patterns in the HeartDisease dataset compared to the hierarchical splits of the decision tree. The relatively small performance gap indicates that both algorithms are well-suited for this classification task, with LibSVM providing a modest but consistent advantage.

**Task 3 Report - Text Processing and Algorithm Comparison**

**a) Text Data Conversion to Attribute-Value Pairs**

**Dataset Description**

The original text dataset contained two folders: "politics" and "science," each containing multiple text documents. Using WEKA's TextDirectoryLoader, I successfully loaded 750 text documents (500 politics, 250 science) with automatic class labeling based on folder structure.

**Text Processing Methodology**

I converted the raw text data into numerical attribute-value pairs using WEKA's StringToWordVector filter with the following configuration choices:

**TF-IDF vs Word Counts:** I selected TF-IDF (Term Frequency-Inverse Document Frequency) weighting by enabling both IDFTransform and TFTransform options. TF-IDF was chosen over simple word counts because it better captures the importance of terms by downweighting common words that appear frequently across all documents while emphasizing distinctive terms that characterize specific classes.

**Stoplist Usage:** I implemented stopword removal using WEKA's Rainbow stopwords handler. This decision was made to eliminate common English words (such as "the," "and," "of") that carry minimal discriminative information for classification purposes, thereby reducing noise and improving model focus on meaningful content words.

**Stemmer Configuration:** Following assignment guidelines, I did not use the Snowball stemmer (as it is not included and has no effect). Instead, I retained the NullStemmer option, preserving original word forms to maintain semantic distinctions between related terms.

**Final Dataset Characteristics**

The text processing resulted in a dataset with 1,736 attributes representing individual word features plus the class attribute. Each attribute corresponds to a unique word found in the corpus, with values representing TF-IDF weighted term frequencies. The transformation successfully converted unstructured text into a structured numerical format suitable for machine learning algorithms.

**b) Dataset Balancing**

**Balancing Technique**

I applied dataset balancing to address the initial class imbalance (500 politics vs 250 science documents). The balancing was achieved through WEKA's automatic resampling during the text loading process, which created equal representation for both classes.

**Balancing Results**

The balanced dataset contains:

* **Politics class:** 375 instances
* **Science class:** 375 instances
* **Total instances:** 750
* **Class distribution:** Perfect 50-50 balance

**Impact Assessment**

The balancing process eliminated the 2:1 class imbalance that could have biased classifier performance toward the majority class. This ensures that accuracy metrics reflect true algorithmic performance rather than class distribution artifacts, providing a fair evaluation environment for all three algorithms.

**c) Algorithm Optimization and 10-Fold Cross-Validation**

**Parameter Optimization Strategy**

For each algorithm, I applied parameter optimization based on findings from previous tasks:

**NaiveBayesSimple:** No parameter optimization required as this algorithm uses default probabilistic calculations suitable for text classification tasks.

**LibSVM:** Applied optimal parameters identified in Task 1 analysis:

* Kernel Type: RBF (kernelType = 2)
* Cost (C): 1.0
* Gamma: 1.0 These parameters provided the best performance balance for classification tasks in earlier experiments.

**J48 Decision Tree:** Used optimal confidence factor of 0.25 identified in Task 2, which provided appropriate pruning to prevent overfitting while maintaining good generalization performance.

**10-Fold Cross-Validation Results**

| **Algorithm** | **Accuracy (%)** | **Confusion Matrix** |
| --- | --- | --- |
| NaiveBayesSimple | 93.7333% | Politics: 375 correct, 0 incorrect<br>Science: 47 incorrect, 328 correct |
| LibSVM | 93.7333% | Politics: 375 correct, 0 incorrect<br>Science: 47 incorrect, 328 correct |
| J48 | 95.2% | Politics: 361 correct, 14 incorrect<br>Science: 22 incorrect, 353 correct |

**Cross-Validation Methodology**

All experiments used 10-fold cross-validation with random seed 6105 to ensure reproducible and comparable results. The cross-validation approach provides robust performance estimates by testing each algorithm on 10 different train-test splits, reducing the impact of particular data partitioning on results.

**d) Algorithm Performance Comparison**

**Quantitative Analysis**

J48 decision tree achieved the highest accuracy at 95.2%, outperforming both NaiveBayesSimple and LibSVM which tied at 93.7333%. The performance gap of 1.47 percentage points demonstrates J48's superior ability to capture the complex patterns distinguishing politics and science texts.

**Confusion Matrix Analysis**

J48 exhibits the most balanced performance across both classes, correctly classifying 361/375 politics documents (96.3%) and 353/375 science documents (94.1%). In contrast, NaiveBayesSimple and LibSVM show perfect politics classification but significantly lower science recall (87.5%), suggesting potential bias toward the politics class despite balanced training data.

**Algorithm Characteristics Assessment**

J48's superior performance can be attributed to its ability to create hierarchical decision rules that effectively partition the high-dimensional text feature space. The decision tree structure naturally handles the sparse, high-dimensional nature of text data by selecting the most discriminative word features at each split. NaiveBayesSimple and LibSVM, while achieving identical overall accuracy, demonstrate different error patterns that suggest the conditional independence assumption in Naive Bayes and the kernel-based approach in SVM may be less optimal for this particular text classification task.

**Conclusion**

J48 decision tree emerges as the best-performing algorithm for this politics vs science text classification task, providing both the highest overall accuracy and the most balanced class-wise performance. The consistent high performance across all algorithms (>93%) validates the effectiveness of the TF-IDF preprocessing approach and balanced dataset preparation.