EX05

- 7.5, 7.9, 7.10
- 8.3, 8.5, 8.7, 8.12, 8.14
- Due 3/24

Zhenhao Zhang
zzh133@u.rochester.edu

7.5

Section 7.2.4 presented various ways of defining negatively correlated patterns. Consider Definition 7.3: "Suppose that itemsets X and Y are both frequent, that is, $sup(X) \geq min\ sup$ and $sup(Y) \geq min\ sup$, where $min\ sup$ is the minimum support threshold. If $(P(X|Y)+P(Y|X))/2 < \varepsilon$, where ε is a negative pattern threshold, then pattern $X \cup Y$ is a **negatively correlated pattern**." Design an efficient pattern growth algorithm for mining the set of negatively correlated patterns.

First, the support values sup(X), sup(Y), and sup(Z) are pre-stored. Its conditional pattern base and subsequent conditional FP-tree are constructed for each frequent item. Then, this process is repeated on each newly generated conditional FP-tree until the FP-tree is empty or only forms a single path. Similar to the FP-growth algorithm, a single path can generate all combinations of its sub-paths, each representing a frequent pattern. However, unlike FP-growth, our goal at this point is to record information on frequent and infrequent patterns. Finally, for each frequent itemset X, we traverse the infrequent itemsets Z that includes X. Define Y = Z \ X, meaning Y comprises elements in Z that are not in X. If Y is a frequent itemset. We calculate ((P(X|Y) + P(Y|X)) / 2 = (sup(Z) / sup(Y) + sup(Z) / sup(X)) / 2) to determine whether X and Y have a negative correlation. Alternatively, the NCP-growth method could also be employed, which similarly aims to identify negatively correlated patterns but might differ in implementation specifics.

7.9

Section 7.5.1 defined a **pattern distance measure** between closed patterns P_1 and P_2 as

$$Pat\_Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|},$$

where $T(P_1)$ and $T(P_2)$ are the supporting transaction sets of P_1 and P_2, respectively. Is this a valid distance metric? Show the derivation to support your answer.

$$Pat\_Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|}$$

We assume that $T(P_1) = (t_1, t_2, t_3, t_4, t_5)$, $T(P_2) = (t_1, t_2, t_3, t_4, t_5)$ where t is the transaction dataset distance. T_1 is $D(P_1, P_2)$

According to the Pat_Dist, which is a valid distance metric, it has the following properities:

(1) $Pat\_Dist(P_1, P_2) > 0, \forall P_1 \neq \forall P_2$

(2) $Pat\_Dist(P_1, P_2) = 0, \quad \forall P_1 = \forall P_2$

(3) $Pat\_Dist(P_1, P_2) = Pat\_Dist(P_2, P_1)$

(4) $Pat\_Dist(P_1 + P_2) + Pat\_Dist(P_2, P_3) \geq Pat\_Dist(P_1, P_3) \quad \forall P_1, \forall P_2, \forall P_3$

Then we assume the variables:

$|T(P_1)| = a$, $|T(P_2)| = b$, $|T(P_3)| = c$, $|T(P_1) \cap T(P_2)| = b_1$, $|T(P_2) - T(P_1) \cap T(P_3)| = b_2$

$|T(P_1) \cap T(P_3)| = c_1$, $|T(P_3) - T(P_1)| = c_2$, $|T(P_1) \cap T(P_2) \cap T(P_3)| = d$

$T(P_2) \cap T(P_3) - |T(P_1) \cap T(P_2) \cap T(P_3)| = d_2$

Since $(T(P_1) \cap T(P_2)) \cup T(P_1) \cap T(P_3)) \subseteq T(P_1)$

$\Rightarrow |T(P_1) \cap T(P_2)| + |T(P_1) \cap T(P_3)| - |T(P_1) \cap T(P_2) \cap T(P_3)| \leq |T(P_1)|$

$\Rightarrow b_1 + c_1 - d_1 \leq a$ ①

$Pat\_Dist(P_1, P_2) + Pat\_Dist(P_2, P_3) \geq Pat\_Dist(P_1, P_3)$

$$\Rightarrow \frac{b_1}{a + b_2} + \frac{c_1}{a + c_2} \leq 1 + \frac{d_1 + d_2}{b_1 + b_2 + c_1 + c_2 - d_1 - d_2}$$

$$\Rightarrow 1 + \frac{d_1 + d_2}{b_1 + b_2 + c_1 + c_2 - d_1 - d_2} \geq 1 + \frac{d_1}{b_1 + b_2 + c_1 + c_2 - d_1} \quad (d_2 > 0)$$

$$\geq 1 + \frac{d_1}{a + b_2 + c_2} = \frac{a + b_2 + c_2 + d_1}{a + b_2 + c_2} \quad ①$$

$$\geq \frac{b_1 + c_1 + b_2 + c_2}{a + b_2 + c_2} = \frac{b_1 + c_2}{a + b_2 + c_2} + \frac{c_1 + b_2}{a + b_2 + c_2}$$

$$\geq \frac{b_1}{a + b_2} + \frac{c_1}{a + c_2}$$

$(a+b_2 \geq b_1,\ c_2 \geq 0)$

$(a+c_2 \geq c_1,\ b_2 \geq 0)$

Thus, the distance between equation is correct.

7.10

Association rule mining often generates a large number of rules, many of which may be similar, thus not containing much novel information. Design an efficient algorithm that **compresses** a large set of patterns into a small compact set. Discuss whether your mining method is robust under different pattern similarity definitions.

To efficiently compress a vast number of patterns into a concise and meaningful subset, we employ a strategy that combines global greedy algorithms with local greedy algorithms. Within this framework, greedy algorithms are utilized to select representative patterns. The global greedy algorithm aims to find representative patterns that can cover all other patterns to the greatest extent possible. However, as the number of frequent patterns increases, the efficiency of global optimization decreases due to its high computational intensity. The local greedy method selects the locally optimal representative pattern at each step as a more efficient alternative. Although it may not achieve the global optimal solution, it has a lower computational complexity when dealing with large datasets.

In practice, we start with the local optimization phase, where the large dataset is divided into several smaller subsets. Local greedy algorithms are applied within these subsets to quickly identify their respective sets of locally optimal representative patterns. These patterns are chosen because they cover most other patterns within their subsets. Then, in the global optimization phase, the representative pattern sets from each subset are combined into a preliminary global pattern set. This set is then finely optimized with a global greedy algorithm to filter out the truly representative global pattern set. This stage may require reevaluating the patterns selected during the local phase to ensure they remain optimal across the entire dataset.

Next, we iteratively refine the merged global pattern set, adjusting representative patterns during each iteration by removing overlapping or less informative patterns and adding new potential representative patterns. This process continues until there is no significant improvement in the quality of the results, indicating convergence.

In the final post-processing stage, the ultimate pattern set undergoes quality assessment, all redundant patterns are removed, and further selection is made based on the final application. By combining global and local greedy algorithms, we ensure the accuracy of the results and enhance processing efficiency, finding an ideal balance between the two. This strategy is particularly suitable for processing large-scale datasets, as relying solely on global greedy algorithms may be impractically costly while depending solely on local greedy algorithms might miss the global optimum. Through careful design and parameter adjustments, this combined strategy provides an excellent compromise between efficiency and accuracy.

8.3

Given a decision tree, you have the option of (a)*converting* the decision tree to rule and then pruning the resulting rules, or (b) *pruning* the decision tree and then converting the pruned tree to rules. What advantage does (a) have over (b)?

Opting to convert a decision tree into rules before proceeding with pruning, as opposed to pruning the tree first and then translating it into rules, confers a distinct advantage centered around the granularity and adaptability of the pruning process. This approach allows each path from the decision tree's root to a leaf to be transformed into an individual rule, offering a more detailed level of pruning. Each rule can be independently assessed and pruned based on its merits, such as accuracy, support, or confidence. This method enables the selective removal of specific conditions within rules that might not significantly enhance prediction accuracy, fostering the creation of more generalized and potentially robust rules. Additionally, converting to rules before pruning provides the flexibility to apply varied pruning criteria and thresholds to each rule, tailoring the optimization to the unique characteristics of individual rules. This could lead to a pruning strategy that is both more nuanced and effective, yielding a rule set that is more accurate and easier to understand. The simplification achieved through this process can enhance the interpretability of the model, as it allows for the elimination of superfluous conditions within rules, rendering each rule more straightforward and comprehensible.

8.5

Given a 5-GB data set with 50 attributes (each containing 100 distinct values) and 512 MB of main memory in your laptop, outline an efficient method that constructs decision trees in such large data sets. Justify your answer by rough calculation of your main memory usage.

An algorithm that can process data in batches without loading the entire dataset is needed to construct decision trees for large datasets under limited main memory conditions. The SLIQ algorithm and its extension SPRINT are designed for this purpose, utilizing attribute value pre-sorting techniques and class lists that associate attribute values with target variables.

Initially, each attribute is pre-sorted in the data preprocessing stage, and the sorted results are stored on disk for efficient sequential access. A class list is also created to maintain the mapping between attribute values and class labels.

In the subsequent decision tree construction stage, the class list is used to determine the optimal split point for each attribute by calculating information gain or the Gini index, without needing to load all data into memory. After identifying the best splitting attribute, the class list is partitioned into subsists, each corresponding to a different attribute value.

Regarding memory management, only the necessary portions of the class list must be loaded into memory when performing splits. Assuming each attribute has 100 distinct values, and the data is evenly distributed, each value represents approximately 1/100 of the data for that attribute. If there are 50 attributes with 100 distinct values each, and an efficient binary representation is adopted, the memory usage for the attribute lists would be approximately 50 * 100 * 8 bytes = 40,000 bytes, or around 0.04MB.

The class list needs to reference all records, so for 5GB of data, assuming a 32-bit integer is used for each record reference, the class list will occupy 5GB / 4 bytes, approximately 1.25 billion references. However, these references are loaded sequentially into memory and not stored all at once, effectively controlling memory usage.

This approach allows decision trees to be efficiently constructed even on a laptop with limited main memory, without the need to load the entire dataset simultaneously.

8.7

The following table consists of training data from an employee database. The data have been generalized. For example, “31 ... 35” for *age* represents the age range of 31 to 35. For a given row entry, *count* represents the number of data tuples having the values for *department, status, age*, and *salary* given in that row.

department	*status*	*age*	*salary*	*count*
sales	senior	31...35	46K...50K	30
sales	junior	26...30	26K...30K	40
sales	junior	31...35	31K...35K	40
systems	junior	21...25	46K...50K	20
systems	senior	31...35	66K...70K	5
systems	junior	26...30	46K...50K	3
systems	senior	41...45	66K...70K	3
marketing	senior	36...40	46K...50K	10
marketing	junior	31...35	41K...45K	4
secretary	senior	46...50	36K...40K	4
secretary	junior	26...30	26K...30K	6

Let *status* be the class label attribute.

(a) How would you modify the basic decision tree algorithm to take into consideration the *count* of each generalized data tuple (i.e., of each row entry)?

The basic decision tree algorithm must be adapted to account for the count of each generalized data tuple: Each tuple's count should be integrated into the computation of the attribute selection metric, such as information gain, and the count must also be considered when determining the most common class among the tuples.

Firstly, when choosing an attribute to split on, the information gain or other metrics for each attribute must be calculated based on their individual contributions and weighted by the count of each tuple. This ensures that the algorithm accounts for the actual influence of each tuple, particularly for those tuples that have generalized a range of instances.

Secondly, when deciding on the class label for each node, we weigh the occurrence of each class by the count of the tuples. This means that a tuple with a larger count will have a greater impact on determining the most common class in the entire dataset. Through this method, we ensure that each decision node in the decision tree accurately reflects the true distribution of the data.

In summary, this modified approach allows the algorithm to process generalized data tuples more accurately and truly represent the characteristics of the original dataset when

building the decision tree. Consequently, the decision tree will be capable of generating a more precise model that can better predict the class of new samples.

(b) Use your algorithm to construct a decision tree from the given data.

The decision tree is:

(salary = 26K...30K:

junior

= 31K...35K:

junior

= 36K...40K:

senior

= 41K...45K:

junior

= 46K...50K (department =secretary:

junior

= sales:

senior

= systems:

junior

= marketing:

senior)

= 66K...70K:

senior)

(c) Given a data tuple having the values *"systems," "26...30,"* and *"46–50K"* for the attributes *department, age*, and *salary*, respectively, what would a naïve Bayesian classification of the *status* for the tuple be?

P(X | senior) = 0;

P(X | junior) = 31 / 113 × 46 / 113 × 20 / 113 = 0.018.

Thus, a naïve Bayesian classification predicts "junior".

8.12

The data tuples of Figure 8.25 are sorted by decreasing probability value, as returned by a classifier. For each tuple, compute the values for the number of true positives (*TP*), false positives (*FP*), true negatives (*TN*), and false negatives (*FN*). Compute the true positive rate (*TPR*) and false positive rate (*FPR*). Plot the ROC curve for the data.

Tuple #	*Class*	*Probability*
1	*P*	0.95
2	*N*	0.85
3	*P*	0.78
4	*P*	0.66
5	*N*	0.60
6	*P*	0.55
7	*N*	0.53
8	*N*	0.52
9	*N*	0.51
10	*P*	0.40

```
# Zhenhao Zhang 3/19/24


import pandas as pd
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Given data
data = pd.DataFrame({
    'Tuple #': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Class': ['P', 'N', 'P', 'P', 'N', 'P', 'N', 'N', 'N', 'P'],
    'Probability': [0.95, 0.85, 0.78, 0.66, 0.60, 0.55, 0.53, 0.52,
0.51, 0.40]
})

# Initialize counters
TP = 0
FP = 0
TN = 0
FN = 0

# Calculate TP, FP, TN, FN for each threshold
results = []
for threshold in data['Probability']:
    for index, row in data.iterrows():
```

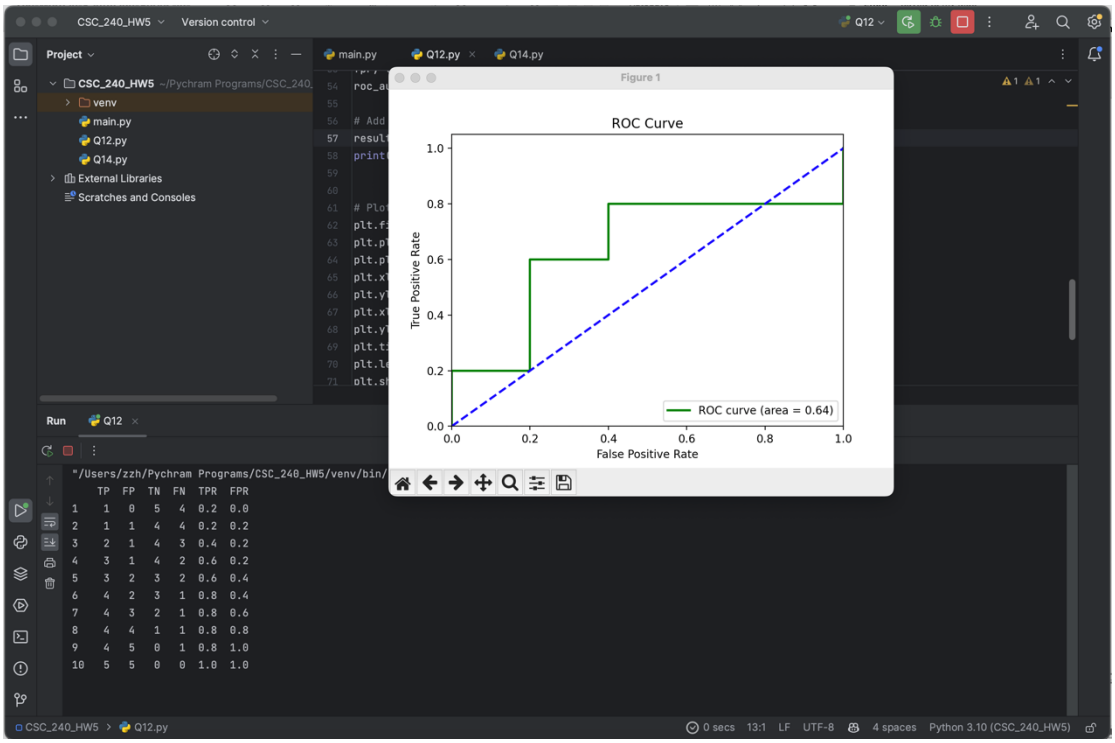
```
        if row['Probability'] >= threshold:
            if row['Class'] == 'P':
                TP += 1
            else:
                FP += 1
        else:
            if row['Class'] == 'N':
                TN += 1
            else:
                FN += 1
    TPR = TP / (TP + FN) if TP + FN > 0 else 0
    FPR = FP / (FP + TN) if FP + TN > 0 else 0

    results.append((TP, FP, TN, FN, TPR, FPR))

    # Reset counters for the next threshold
    TP = 0
    FP = 0
    TN = 0
    FN = 0

# Convert to DataFrame for easier handling
results_df = pd.DataFrame(results, columns=['TP', 'FP', 'TN', 'FN',
'TPR', 'FPR'])

# Calculate the ROC curve and AUC
y_true = data['Class'].map({'P': 1, 'N': 0}).values
y_scores = data['Probability'].values
fpr, tpr, thresholds = roc_curve(y_true, y_scores)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='green', lw=2, label='ROC curve (area
= %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

```
results_df.index += 1
print(results_df)
```

Output:

	TP	FP	TN	FN	TPR	FPR
1	1	0	5	4	0.2	0.0
2	1	1	4	4	0.2	0.2
3	2	1	4	3	0.4	0.2
4	3	1	4	2	0.6	0.2
5	3	2	3	2	0.6	0.4
6	4	2	3	1	0.8	0.4
7	4	3	2	1	0.8	0.6
8	4	4	1	1	0.8	0.8
9	4	5	0	1	0.8	1.0
10	5	5	0	0	1.0	1.0

ROC curve:

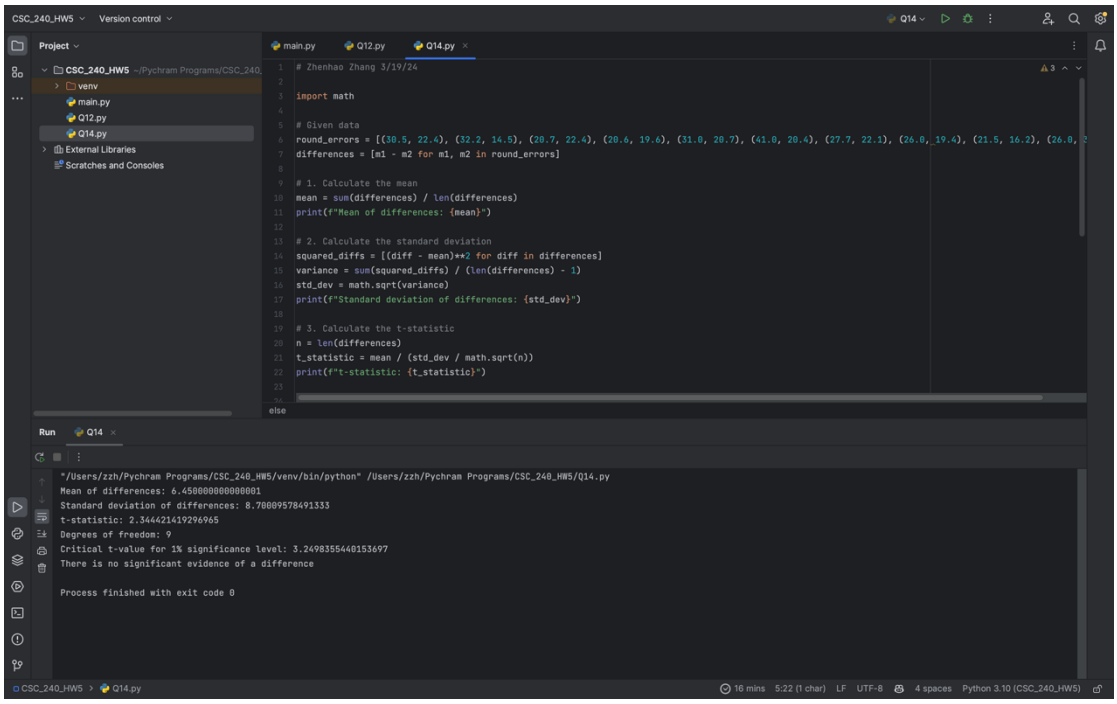


8.14

Suppose that we want to *select between two prediction models*, M_1 and M_2. We have performed 10 rounds of 10-fold cross-validation on each model, where the same data partitioning in round i is used for both M_1 and M_2. The error rates obtained for M_1 are 30.5, 32.2, 20.7, 20.6, 31.0, 41.0, 27.7, 26.0, 21.5, 26.0. The error rates for M_2 are 22.4, 14.5, 22.4, 19.6, 20.7, 20.4, 22.1, 19.4, 16.2, 35.0. Comment on whether one model is significantly better than the other considering a significance level of 1%.

```
import math

# Given data
round_errors = [(30.5, 22.4), (32.2, 14.5), (20.7, 22.4), (20.6,
19.6), (31.0, 20.7), (41.0, 20.4), (27.7, 22.1), (26.0, 19.4), (21.5,
16.2), (26.0, 35.0)]
differences = [m1 - m2 for m1, m2 in round_errors]

# 1. Calculate the mean
mean = sum(differences) / len(differences)
print(f"Mean of differences: {mean}")

# 2. Calculate the standard deviation
squared_diffs = [(diff - mean)**2 for diff in differences]
variance = sum(squared_diffs) / (len(differences) - 1)
std_dev = math.sqrt(variance)
print(f"Standard deviation of differences: {std_dev}")

# 3. Calculate the t-statistic
n = len(differences)
t_statistic = mean / (std_dev / math.sqrt(n))
print(f"t-statistic: {t_statistic}")

# 4. Degrees of freedom
df = n - 1
print(f"Degrees of freedom: {df}")

# 5. Find the critical value
import scipy.stats as stats
critical_value = stats.t.ppf(0.995, df)  # 0.5% critical value for
two-tailed test
print(f"Critical t-value for 1% significance level:
{critical_value}")

# 6. Compare the t-statistic and critical value
```

```
if abs(t_statistic) > critical_value:
    print("There is a significant difference")
else:
    print("There is no significant evidence of a difference")
```



Mean of differences: 6.450000000000001

Standard deviation of differences: 8.70009578491333

t-statistic: 2.344421419296965

Degrees of freedom: 9

Critical t-value for 1% significance level: 3.2498355440153697

There is no significant evidence of a difference

The value for probability 0.005 and 9 degrees of freedom is 3.2498355440153697 ≈ 3.25. Also, given that -3.25 < 2.344 < 3.25, we accept the null hypothesis. Thus, the two models are not different at a significance level 0.01.