Software Analytics - LTAT.05.008

Homework HW5: transformation and mining

Submission deadline: Homework solutions must be submitted within seven days, i.e., not later than on the following Monday, at 23:59 hours.

Late submission policy:

- 50% of the total marks deducted for submission up to 24 hours late
- 100% of the total marks deducted for submission more than 24 hours late

Maximum amount of points is ten (10).

Knowledge Discovery

During three lab sessions, you will walk through the standard data mining workflow and learn how to apply it to real problems in the context of software analytics.

We have chosen four real problems that are still interesting for the Software Engineering research community in order to show how the knowledge discovery process can help. These problems are:

- Estimating story points from User Stories [1, 4]
- Assignment of developers to tasks
- Exploring the GitHub social network [3]

To provide a better understanding of how to solve these problems, we have broken the solutions down into three steps that correspond to three labs sessions:

- KD1: data selection and pre-processing
- KD2: data transformation and data mining
- KD3: evaluation and interpretation
KD2: data transformation and data mining

During the first lab you will walk through some common tasks related to data transformation and data mining. At the end of the lab, it is expected you will be able:

- to apply the required data transformations to run data mining algorithms
- to apply data mining algorithms

Preparation

- Background information on the necessary topics for this homework can be found in the lecture slides.
- Familiarize yourself with Python, IPython Notebook, and Pandas.
- In addition, please consider following the Python Tutorial under Additional Links.

Homework

Task 1 Estimating Story Points

The main goal of this task is to apply a prediction model to get Story Points estimates for the issues used in the Spring XD project. We can treat this problem as a classification task, in which you will try to predict the estimates (in Story Points) for each issue. To train the classifier, you will use features extracted from not only the issues (e.g. summary, description, priority) but also the developer (e.g. reputation, number of comments, workload).

This Task consist of two parts. In the first part (Task 1.1) you will transform the data by adding new features whereas in the second one (Task 1.2) you will apply the corresponding algorithm to classify the issues.

Task 1.1 Transformation. Adding new features.

In this part, you will add new features to the dataset that you have created during the previous lab (KD1 - Task 1). Recall that this dataset contains the data about the issues of the Spring XD project. We will use the cleaned version of this dataset: "issues-xd-cleaned.csv"

If you check the fields in the dataset, you will find a feature called "Story Points". We will use this field as the prediction target and the remaining features + new ones to train the classifier.

The new features will be created as a result of transforming the original features in the dataset.

Your task is to create a modified dataset that contains the following additional features:
Textual features:

- Context. Put together the summary and description fields in a new feature named context.

Example:

<table>
<thead>
<tr>
<th>key</th>
<th>summary</th>
<th>description</th>
<th>context</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>FIX composed job error message</td>
<td>As a user, I'm trying to compose a job just with one definition; however, I'm getting the following error message, which could be misinterpreted.{code}xd:&gt;job create salsa --definition timestampfileSuccessfully created job 'salsa'xd:&gt;job create foo --definition &quot;salsa</td>
<td></td>
</tr>
</tbody>
</table>

In [ ]:

```
# TODO: write code here
# ...
```
Code. Since developers often include not only a description of the issue in natural language but also code snippets to describe particular situations, it is convenient to analyze the description in natural language and the code snippet separately. This is because the language used in the blocks of code might have different meanings from those found in the natural language descriptions.

Example:

<table>
<thead>
<tr>
<th>key</th>
<th>context</th>
<th>context_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 XD-3683</td>
<td>Fix composed job error message. As a user, I'm trying to compose a job just with one definition; however, I'm getting the following error message, which could be misinterpreted.</td>
<td>{code} xd:&gt;job create salsa --definition timestampfile Successfully created job 'salsa' xd:&gt;job create foo --definition &quot;salsa</td>
</tr>
</tbody>
</table>

In [ ]:

```
# TODO: write code here
# ...
```
• N-grams. Use Term Frequency - Inverse Document Frequency (TF-IDF) normalization to extract uni-grams and bi-grams from the context.

NB: You might use the class TfidfVectorizer from sklearn to get the n-grams using tf-idf. Here is an example:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import scipy.sparse as sp

corpus = ['This is the first document.',
          'This is the second document.',
          'And the third one.',
          'Is this the first document?',
]

# defining the desired vectorizer
vectorizer = TfidfVectorizer(ngram_range=(1, 2), analyzer='word', min_df=.01, stop_words='english')
# apply the vectorizer to the target and get the result
x = vectorizer.fit_transform(corpus)

# getting the name of the n-grams generated
print vectorizer.get_feature_names()

# getting the matrix
print x.toarray()
```
Issue features:

- Discussion time. The result of subtracting the resolution date from the creation date.
  \[ \text{Discussion time} = \text{created} - \text{resolution.date} \]

Developers’ features:

- Reputation. Developer reputation has been used in several studies as a way to characterize the role played by developers in software projects. The reputation of a developer \( D \) is calculated as the ratio of the number of issue reports in the dataset that have been both opened and fixed by the developer to the number of issue reports opened by the developer plus one.
  \[
  \text{Reputation}(D) = \frac{|\text{opened}(D) \cap \text{fixed}(D)|}{|\text{opened}(D)| + 1}
  \]

To calculate the reputation, you will have to inspect the dataset and think about:

- What field(s) do(es) determine an opened issue?
- What field(s) do(es) determine a fixed issue?

Total work capacity (in story points): The total number of story points that have been completed by the developer during the project.

Putting everything together

Make sure you put all the features together in a new dataframe (the original features + the new ones). There are many ways to do this. For example, you can use the merge function provided by pandas that joins two data frames in a SQL-style.

```python
import pandas as pd
merged_df = pd.merge( old_df, new_df, on='key', how='left')
```
Task 1.2 Data mining. Training a classifier using SVM.

In this task, you will train a classifier using Support Vector Machines (SVMs).

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

First, we will split the data into a training and testing set. We can use the following code:

```python
from sklearn.cross_validation import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(features, target)
```

Now, divide the dataset into training and testing set:

```python
# TODO: write code here
# ...
```

Then, we can train the classifier using sklearn. The algorithm take as input two arrays: an array X of size [n_samples, n_features] holding the training samples, and an array y of class labels (strings or integers), size [n_samples]:

```python
from sklearn import svm
X = [[0, 0], [1, 1]] # features example
y = [0, 1] # target example

clf = svm.SVC()
clf.fit(X, y)
```

Now, train your classifier using the Xtrain and ytrain created in the previous step.

```python
# TODO: write code here
# ...
```

After being fitted, the model can then be used to predict new values:

```python
clf.predict([[2., 2.]])
```

Now, predict the story points for the issues in the Xtest set.
Please answer the following question: **What is the predicted effort (in story points) for the first issue in the test dataset Xtest[0]??**

**Task 2 Assigning issues to developers**

In agile software development, user story allocation is often based on self-assignment. That is, developers choose the user stories that they will develop during the sprint according to their own preferences and experience. Industry practices give some evidence to support this method of task allocation but how this takes place is not completely clear yet. As far we know, developers apply different strategies for self-assigning different types of issues (new features, enhancements, bug fixation). However, applying these strategies to determine what issue develop can be difficult for non-experienced developers. The goal of this project is to use features about the developers to recommend issues to developers. To this end, you will use clustering techniques to provide the recommendations.

During this task, you will use the jira dataset of the Spring XD project. In task 2.1 you will transform the data in order to let it ready to apply a clustering algorithm. In task 2.2, you will apply the clustering technique.

**Task 2.1. Transformation**
The problem of recommending issues to developers can be formulated as follows.

Let $I^{m \times q}$ be a matrix that represents all the issues-related info, where $m$ represents the number of issues (tasks, stories, bugs, etc.) for a given project, and $q$ the total number of features related to the issues.

<table>
<thead>
<tr>
<th>Issues’ features</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Issue 2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Issue 3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Issue 4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Issue 5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$I^{m \times q}$

The matrix in the example above is filled with 1 and 0 just to make easier the example. These values can be integers or real numbers. **If you have nominal values, you will probably want to create dummy variables for that.** [https://towardsdatascience.com/the-dummys-guide-to-creating-dummy-variables-f21faddb1d40](https://towardsdatascience.com/the-dummys-guide-to-creating-dummy-variables-f21faddb1d40)

Now, create the matrix $I$ using your dataset about the JIRA issues of the Spring XD project.

In [ ]:

```python
# TODO: write code here
# ...
```

Since we have the info about the developers who completed certain issues, we can also define a matrix of completed issues $C^{n \times m}$. Here, each element in the matrix $c_{ij}$ represents whether the $j$-issue has been completed by the $i$-developer.

$$c_{ij} = \begin{cases} 
1 & \text{if the } j\text{-task was completed by the } i\text{-developer} \\
0 & \text{otherwise}
\end{cases}$$

<table>
<thead>
<tr>
<th></th>
<th>Issue 1</th>
<th>Issue 2</th>
<th>Issue 3</th>
<th>Issue 4</th>
<th>Issue 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Dev 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Dev 3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$C^{n \times m}$

Now, create the matrix $C$ using your dataset about the JIRA issues of the Spring XD project.
Once the matrices $I^{m \times q}$ and $C^{n \times m}$ have been defined, it is possible to get another matrix $A^{n \times q}$ given by the product of $C$ and $I$. This matrix represents the developer-enhanced transactions regarding the tasks they completed.

$$C^{n \times m} \times I^{m \times q} = A^{n \times q}$$
Task 2.2. Getting recommendations.

Clustering is a data mining technique that groups together a set of items having similar characteristics. Applying clustering algorithms such as k-means on matrix $A$, allow us to discover clusters of developers. The k-means algorithm can partition this space into groups of transactions that are close to each other based on a measure of distance or similarity among the vectors.

```python
from sklearn.cluster import KMeans
import numpy as np
X = np.array([[1, 2], [1, 4], [1, 0],
              [4, 2], [4, 4], [4, 0]])
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

print(kmeans.labels_)

print(kmeans.predict([[0, 0], [4, 4]]))

print(kmeans.cluster_centers_)
```

We can use the transaction clusters to obtain recommendations. Since these clusters can contain many transactions, it is common to represent each cluster through their centroid or mean vector. The dimension value for each issue feature in the mean vector is computed by finding the ratio of the sum of the issue feature weights across the transactions to the total number of transactions in the cluster. Thus, each centroid dimension value provides a measure of its significance in the cluster.

The issue features in the centroid can be sorted according to these weights and lower weight features can be filtered out. The resulting set of features-weight pairs can be viewed as an **aggregate developer profile** representing the interests or behavior of a significant group of developers. The following figure shows an example of an aggregate profile.

The aggregate developer profile can be easily retrieved by using the line
Now, apply the k-means algorithm to the matrix $A$.

In [ ]:
```
# TODO: write code here
# ...
```

Please answer the following questions:

**What number of clusters (k) have you chosen? Why?**

In [ ]:
```
```

This aggregate representation can be used directly in applications such as recommender systems: given a task we can measure the similarity of their features to the discovered profiles, and recommend the task issues to a group of developers.
Task 3. Exploring the GitHub social network

In this task, we will use a property graph for the purpose of modeling GitHub data as an interest graph by way of a Python package called NetworkX.

A property graph is a data structure that represents entities with nodes and relationships between the entities with edges. Each vertex has a unique identifier, a map of properties that are defined as key/value pairs, and a collection of edges. Likewise, edges are unique in that they connect nodes, can be uniquely identified, and can contain properties.

Task 3.1. Creating the property graph

In the previous homework, you have extracted the info about all the contributors of the spring-xd project as well as the repositories in which they are involved (in addition to the spring-xd project). Using this dataset, create a directed graph representing the following relationship:

![Diagram showing a directed graph with a user node connecting to a repo node with an 'contributes' edge]

In this directed graph, the nodes can be: the contributors of the spring-xd project (users) or the repositories in which the users have contributed (repos). The edges in the graph determine whether a user has contributed to the repository or not.

Hint. Take care to ensure that user and repo nodes do not collide by appending their type. For example, a user can be called as 'spring-xd' and a repo can have exactly the same name. Github doesn't have restriction about it. Thus, we can distinguish the user from the repo by just adding a string to the name. Then, 'spring-xd(user)' will refer to the user and 'spring-xd(repo)' to the repository.

Here is an example of how to use networkx:

To install the networkx package, you can use pip install networkx

```python
import networkx as nx

g = nx.DiGraph()  # create a new directed graph

# create a node
    g.add_node( repo.name + '(repo)',  # name of the node
type='repo',  # type of the node
    lang=repo.language,  # properties of the node using key=value
    owner=user.login  # you can add many properties as you want
)

# create an edge
    g.add_edge( contributor.login + '(user)',  # from user
repo.name + '(repo)',  # to repo
type='contributes'  # type of edge
)
```

Now, we will transform the previous directed graph into a graph that have more emphasis on the users (developers) than in the repositories.

Your task is to write an algorithm to transform the previous graph into one that represents the following relationship:

In this undirected graph, nodes can be only contributors (users), the edges represent whether two users have repositories in common or not, and the weights of the edges show the number of repositories.

Please answer the following question:

**How much has the new graph been reduced? Measure the size of the graph in terms of number of nodes, edges and average degree.**

You can use the method `nx.info()` of the package `networkx` to get the number of nodes, edges and average degree:

```python
import networkx as nx

# h is the graph
print nx.info(h)
```

Please answer the following question:

```python
# TODO: write code here
# ...
```
Task 3.2. Community detection.

In this task, you will detect communities on the Github social network. Using community detection algorithms, we can break down a social network into different potentially overlapping communities. Communities are groups of densely connected nodes with few connections to nodes outside of the group.

While there is no community detection method in the NetworkX library, a good samaritan has written a community detection library built on top of NetworkX.

https://bitbucket.org/taynaud/python-louvain

To install this package:

    pip install python-louvain

The library is easy to use and allows to perform community detection on an undirected graph in less than 3 lines of code:

    import community

    # G is a graph created using networkx
    # we have to use the parameter weight to specify the field that we want to use as weight
    parts = community.best_partition(G, weight='common')
    values = [parts.get(node) for node in G.nodes()]

Your task to detect the communities in the transformed graph that you have created before. Remember this graph is undirected and it models the following relationship:

In [ ]:

    # TODO: write code here
    # ...

Please answer the following question:

How many communities has been detected?

In [ ]:

    # TODO: write your answer or code here
    # ...
[OPTIONAL] Graph visualization

NB This part is for your own learning and you will not obtain points for this task.

Graph visualisations are a powerful tool to convey the content of a graph. They can highlight patterns, and show clusters and connections. To visualize the graph and the communities you have created, you can use networkx and matplotlib. Here is an example on how to visualize a graph with communities:

```python
import community
import networkx as nx
import matplotlib.pyplot as plt

%matplotlib inline

#first compute the best partition
partition = community.best_partition(G, weight='common')

# visualizing the partitions
size = float(len(set(partition.values())))
# create a layout, there are many different layouts
pos = nx.spring_layout(G)
count = 0.
for com in set(partition.values()) :
    count = count + 1.
    list_nodes = [nodes for nodes in partition.keys()
                   if partition[nodes] == com]
    nx.draw_networkx_nodes(G, pos, list_nodes, node_size = 40,
                           node_color = str(count / size))

plt.axis("off")

nx.draw_networkx_edges(G, pos, alpha=0.5)
plt.show()
```

In [ ]:

```
# TODO: write your answer or code here
# ...
```
Submission

The solution to this homework must be submitted in a zip file via the course web-page selecting Lab 5. The zip file must contain:

- Your name
- The code scripts for all the tasks. Ideally, you will complete the IPython Notebook provided. If you are using another language, you must submit the code/scripts used.
- The dataset files you have created in CSV format (if any).
- The answers to the questions. If you are using the IPython Notebook, you can write the answers in the cells. If you are using another language, please submit a PDF file with your answers.

Grading

You can get up to 10 points for this lab. The grading is as follows:

- 1p if you use your own dataset extracted during the previous homework
- Task 1 - Estimating Story Points (3 points in total)
  - Task 1.1 - Data transformation. Adding new features (1.5p)
  - Task 1.2 - Data mining. Training a classifier using SVM (1.5p)
- Task 2 - Assigning issues to developers (3 points in total)
  - Task 2.2 - Transformation (1.5p)
  - Task 2.3 - Getting recommendations (1.5p)
- Task 3 - Exploring the GitHub social network (3 points in total)
  - Task 3.1 - Creating the property graph (1.5p)
  - Task 3.2 - Community detection (1.5p)

Additional links

- Python tutorial - https://docs.python.org/2/tutorial/ (https://docs.python.org/2/tutorial/)

References


