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 issue reports [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

**KD3: evaluation and interpretation**

During the first two labs (KD1 and KD2) you walked through some common tasks related to data extraction, transformation and data mining. At the end of the lab, it is expected you will able to evaluate and interpret common problems in the context of software analytics.

**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 Evaluating the effort predictions (5 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 have used features extracted from JIRA. In this homework, you will evaluate and interpret the results of your predictions.

Task 1.1 (0.5 points)

Evaluate the accuracy of the model that you have created during HW5

1. Split the dataset about issue reports "issues-xd.csv" into train/test sets.
2. Train the classifier as you have done in HW5 using the training set.
3. Calculate the accuracy of the model on the test dataset.
In [1]:

# Read the dataset
import pandas as pd

dataset = pd.read_csv("issues-xf.csv")

# Add new features
context = dataset[['key', 'summary', 'description']].copy()
context["context"] = context["summary"] + ". " + context["description"]

# CLEAN!
context = context[~pd.isnull(context["context")]

# Separate natural language and the code in context
import re
for ix, line in context.iterrows():
    m = re.search('{code}(.*){code}', line['context'], flags=re.DOTALL)
    if m:
        context.loc[ix, 'context_code'] = line.context[m.start():m.end()]
        context.loc[ix, 'context'] = line.context[:m.start()] + line.context[m.end():]
    else:
        context.loc[ix, 'context_code'] = ""
        context.loc[ix, 'context'] = re.sub(r"\s+", " ", context.loc[ix, 'context'])
        context.loc[ix, 'context_code'] = re.sub(r"\s+", " ", context.loc[ix, 'context_code'])

# Calculate text features
def get_text_features(ctx):
    import sklearn.feature_extraction.text
    from sklearn.feature_extraction.text import TfidfVectorizer
    import scipy.sparse as sp
    import numpy as np

    v = TfidfVectorizer(ngram_range=(1, 2), analyzer='word', min_df=.01, max_df=.1, stop_words=None)
    x = v.fit_transform(ctx["context")
    y = v.fit_transform(ctx["context_code")
    textfeatures = sp.hstack((x, y))
    return textfeatures

textfeatures = get_text_features(context)

# Calculate Discussion time
def get_discussiontime(dataset):
    h = dataset.copy()
    h["discussiontime_hours"] = (pd.to_datetime(h["resolutiondate")] - pd.to_datetime(h["cr
    return h[['key', 'discussiontime_hours")]

dtime = get_discussiontime(dataset)

# CLEAN!
dtime = dtime[~pd.isnull(dtime["discussiontime_hours")])]

from __future__ import division

# Calculate reporter reputation
def get_reputation(developer, dataset):
    pass
Return the reputation of a developer in a project dataset

opened = len(dataset['creator'] == developer)
opened_and_fixed = len(dataset[(dataset['creator'] == developer) & ((dataset['status'] == 'Done') | (dataset['status'] == 'Close') | (dataset['status'] == 'Resolved') | (dataset['status'] == 'Accepted')) & (dataset['assignee'] == developer)])
return opened_and_fixed/(opened+1)

def get_reputations(dataset):
    """
    Returns the reputation of each developer involved in the dataset
    """
    import numpy as np

devs = dataset['creator'].unique()
devs = np.append(devs, dataset['assignee'].unique())
devs = np.append(devs, dataset['reporter'].unique())

    # remove duplicates
    #print(devs)
devs = np.unique(devs)

    print("Total number of devs: ", len(devs))

    reputations = []
    for d in devs:
        reputations.append(get_reputation(d, dataset))

    reputations_df = pd.DataFrame({"developer": devs, "reputation": reputations})
    return reputations_df

reputations = get_reputations(dataset)

# Total number of story points that the developer have done during all the project

def get_dev_total_storypoints(dataset):
    totalsp = dataset[['assignee', 'story points']].groupby('assignee').sum()

    totalsp = totalsp.reset_index()
totalsp.columns = ['developer', 'total_storypoints']

    # Normalize by the total number of story points
    total = sum(totalsp['total_storypoints'])
totalsp['total_storypoints'] = totalsp['total_storypoints'] / total

    return totalsp

#CLEAN!
FIBONACCI_CARDS = [0.5, 1, 2, 3, 5, 8, 13, 20, 40]
dataset = dataset[dataset['story points'].isin(FIBONACCI_CARDS)]

workcapacity = get_dev_total_storypoints(dataset)
workcapacity.sort_values(by='total_storypoints', ascending=False).head()
## select the features
```
FEATURES = ['reputation',
             'discussiontime_hours',
             'total_storypoints',
             'creator',
             'reporter',
             'watches',
             'issuetype',
             'priority',
             'story points',
             'context',
             'context_code',
             ]
```

selected_features = merged[ FEATURES ]

# fill NAs
```
selected_features = selected_features.fillna(value={'context': '',
                                                    'context_code': '',
                                                    'reputation': 0,
                                                    'discussiontime_hours': 0,
                                                    'total_storypoints': 0})
```

## create dummies for nominal features
dummies = pd.get_dummies(selected_features,
columns=['creator', 'reporter', 'issuetype', 'priority'], sparse=False)

# I remove these features because they will be transformed into n-grams
dummies.drop([ 'context', 'context_code'], axis=1, inplace=True)

# Get the n-grams from the text
text_features = get_text_features(selected_features[['context', 'context_code']])

('Total number of devs: ', 158)

In [2]:
dummies.head()

Out[2]:

<table>
<thead>
<tr>
<th>reputation</th>
<th>discussiontime_hours</th>
<th>total_storypoints</th>
<th>watches</th>
<th>story points</th>
<th>creator_a_ayya</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>0.318182</td>
<td>0.011111</td>
<td>0.02832</td>
<td>1.0</td>
<td>8.0</td>
</tr>
<tr>
<td>3</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.0</td>
<td>5.0</td>
</tr>
<tr>
<td>4</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

5 rows × 291 columns
In [3]:

# create a sparse matrix with all the features together
import scipy.sparse as sp
import numpy as np

X = sp.hstack((text_features, dummies))
y = dummies['story points']

print X.shape
print y.shape

(3212, 870)
(3212L,)

In [4]:

# Splitting the dataset into TRAIN/TEST
from sklearn.cross_validation import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score

Xtrain, Xtest, ytrain, ytest = train_test_split( X , y, test_size=.3)

print Xtrain.shape
print Xtest.shape

# Training the SVM
clf = svm.SVC()
clf.fit(Xtrain, ytrain)

# Getting the predictions for the test set
predictions = clf.predict(Xtest)

# Calculating the accuracy
accuracy = accuracy_score(ytest, predictions)

print 'The accuracy for the model is', accuracy

C:\Anaconda2\envs\gl-env\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

(2248, 870)
(964, 870)
The accuracy for the model is 0.454356846473

**Task 1.2 (0.5 points)**

Perform a sanity check about the predictions obtained

1. Calculate the accuracy of a model that makes random predictions.
2. Calculate the accuracy of a model that makes predictions based on the most common value.
In [5]:

```python
import random

# random prediction model
def get_random_predictions(length):
    FIBONACCI_CARDS = [0.5, 1, 2, 3, 5, 8, 13, 20, 40]

    rs = []
    for i in range(0, length):
        rs.append(random.choice(FIBONACCI_CARDS))

    return rs

predictions_random = get_random_predictions(len(ytest))

# Calculating the accuracy
from __future__ import division
accuracy = len(ytest[(ytest == predictions_random)]) / len(ytest)
print 'The accuracy for the RANDOM model is', accuracy

The accuracy for the RANDOM model is 0.113070539419
```

In [6]:

```python
# random prediction model
from scipy import stats
from __future__ import division

mode = stats.mode(ytest, axis=None)[0][0]

accuracy = len(ytest[(ytest == mode)]) / len(ytest)
print 'The accuracy for the MOST COMMON value is', accuracy

The accuracy for the MOST COMMON value is 0.245850622407
```

**Task 1.3 (0.5 points)**

Compute problem-specific metrics

1. Calculate the MAE of the prediction model created in HW5.
2. Calculate the SA of the prediction model created in HW5.

In [7]:

```python
from sklearn.metrics import mean_absolute_error
# Getting the predictions for the test set for the SVM
predictions = clf.predict(Xtest)
mae = mean_absolute_error(ytest, predictions)

print 'MAE for svm model:', mae

MAE for svm model: 1.52385892116
```
In [8]:

```python
from sklearn.metrics import mean_absolute_error

# Get MAE for the random predictions
mae_r = mean_absolute_error(ytest, predictions_random)

print('MAE for svm model:', mae

SA = (1 - (mae/mae_r))*100
print('SA for svm model:', SA

MAE for svm model: 1.52385892116
SA for svm model: 81.8046695981
```

**Task 1.4 (1.5 points)**

Perform hyperparameter tuning

1. Choose a k-fold strategy or a holdout strategy. Using that strategy, learn the best parameters for the
classification model and evaluate the model in terms of MAE and SA.

In [ ]:

```python
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn import SVC

# Split the dataset in two equal parts
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.5, random_state=0)

# Set the parameters by cross-validation
parameter_candidates = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                          'C': [1, 10, 100, 1000]},
                        {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

# Create a classifier object with the classifier and parameter candidates
clf = GridSearchCV(SVC(), param_grid=parameter_candidates, cv=2, n_jobs=4, scoring='accuracy')

# Train the classifier on data1's feature and target data
clf.fit(Xtrain, ytrain)

# View the accuracy score
print('Best score for Xtest:', clf.best_score_)

# View the best parameters for the model found using grid search
print('Best C:', clf.best_estimator_.C)
print('Best Kernel:', clf.best_estimator_.kernel)
print('Best Gamma:', clf.best_estimator_.gamma)

Fitting 2 folds for each of 12 candidates, totalling 24 fits

In [ ]:

# Sanity check
# Apply the classifier trained using Xtrain to Xtest, and view the accuracy score
clf.score(Xtest, ytest)
```
In [ ]:

```python
# Train a new classifier using the best parameters found by the grid search
svm.SVC(C=clf.best_estimator_.C,
         kernel=clf.best_estimator_.kernel,
         gamma=clf.best_estimator_.gamma).fit(Xtrain, ytrain).score(Xtest, ytest)
```

**Task 1.5 (2 points)**

1. Compare the performance in terms of accuracy, MAE, and SA of the model that uses the best parameters with the performance of your first model (without hyperparameter tuning). **Which model have shown the best performance?**

In [ ]:

```python
# Splitting the dataset into TRAIN/TEST
from sklearn.cross_validation import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=.3, random_state=0)

# Training the SVM
from sklearn import svm

# here the parameters found:
clf = svm.SVC(C=clf.best_estimator_.C,
             kernel=clf.best_estimator_.kernel,
             gamma=clf.best_estimator_.gamma)
clf.fit(Xtrain, ytrain)

# Getting the predictions for the test set
predictions = clf.predict(Xtest)

# Calculating the accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(ytest, predictions)
print 'The accuracy for the model is:', accuracy

from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(ytest, predictions)
print 'MAE for svm model:', mae
SA = (1 - (mae/mae_r))*100
print 'SA for svm model:', SA
```

**Task 2 Evaluating the recommender system (5 points)**

In this homework, you will evaluate the performance of the recommender system you have created during HW5.
In [9]:

""
Training the recommender
""

import pandas as pd

issues = pd.read_csv("issues-xd.csv")

# features to consider (pre-processing steps)
issues["components"] = issues["components"].apply(lambda x: str(x).split(';') if pd.notna(x) else [])
issues["n_components"] = issues["components"].apply(lambda x: len(x))
issues["versions"] = issues["versions"].apply(lambda x: str(x).split(';') if pd.notna(x) else [])
issues["n_versions"] = issues["versions"].apply(lambda x: len(x))

issues["time_spent"] = (pd.to_datetime(issues["resolutiondate"]) - pd.to_datetime(issues["created"])).dt.total_seconds()

# cleaning steps
issues = issues[~issues.assignee.isnull()]

# select features
df = issues[['key', 'story points', 'priority', 'watches', 'issuetype', 'components', 'n_components']]
df = pd.get_dummies(df, columns=['priority', 'issuetype'])
dummy = pd.get_dummies(df["components"].apply(pd.Series).stack()).sum(level=0)
df = pd.concat([df, dummy], axis=1)
dummy = pd.get_dummies(df["versions"].apply(pd.Series).stack()).sum(level=0)
df = pd.concat([df, dummy], axis=1)
df = df.drop(columns=['components', 'versions'])
df = df.fillna(0)

# create matrix I
I = df.copy()

# create matrix C
C = issues[['assignee', 'key']].copy()
C['times'] = 1

C = C.pivot(index="assignee", columns="key", values="times")
C = C.fillna(0)

I = I.set_index("key")
I = I.sort_index()
print C.shape
print I.shape

# multiply CxI
A = C.dot(I)
print A.shape

# Have a look at the result:
A.head()

(31, 2469)
(2469, 60)
(31, 60)
from sklearn.cluster import KMeans
import numpy as np

## HERE THE TUNNING SHOULD BE DONE LIKE IN TASK 1.4

k = 21
kmeans = KMeans(n_clusters=k, random_state=0).fit(A)
print(kmeans.labels_)
print(kmeans.cluster_centers_)

[  0  8 15 18 15 17  3 15 12  2 20  1  0 15 14 16  6 11  5  7 19  4 10 15 13 15 10  9 15  0  0]
[[ 25.75 11.25  3.25 ...,  0.  0.  0.  ]
 [ 957.  572.  142.  ...,  3.  5.  0.  ]
 [ 511.2 330.  69.  ...,  0.  4.  0.  ]
 ...,
 [ 31.  16.  4.  ...,  0.  0.  0.  ]
 [ 306.  118.  25.  ...,  0.  0.  0.  ]
 [ 612.  340.  91.  ...,  1.  4.  3.  ]]

# full list of developers
devs = issues['assignee'].unique()

# actual developers
actual = issues[['assignee', 'key']]

# getting the recommended clusters (predictions) for all the issues in the dataset
preds = kmeans.predict(I)
from scipy.spatial import distance

actual = issues[['assignee', 'key']]
preds = kmeans.predict(I)

# create a dataframe of developers and their corresponding cluster numbers
clustersdev = pd.DataFrame({'dev': devs, 'cluster': kmeans.labels_})
# create a dataframe of issues and their corresponding cluster numbers
predictions = pd.DataFrame({'key': issues['key'], 'prediction': kmeans.predict(I)})

result = pd.DataFrame()

# for each issue in the dataset
for ix, iss in actual.iterrows():

    # get the predicted cluster number
    predi = predictions[predictions['key'] == iss.key]['prediction'].values

    # get the centroid of the predicted cluster
    centroid = kmeans.cluster_centers_[predi[0]]

    # get the issue representation as a vector of features (point)
    datapoint = I.loc[iss.key, :].reshape(1, -1)

    # distance from the centroid to the point
    dist = distance.euclidean(datapoint, centroid)

    # get the recommended (predicted) developer
    predicted_devs = clustersdev[clustersdev['cluster'] == predi[0]]['dev'].values

    # append the result to a dataframe
    result = result.append(pd.DataFrame({'key': iss.key,
                                          'actual': iss.assignee,
                                          'predicted': predicted_devs,
                                          'distance': dist}), ignore_index=True)

# as a result, we get a dataframe with <issue, actual_developer, distance (rank), predicted_developer>
result.head()
# sort the recommendations (predictions) according to the distance value to make a rank
sorted_predictions = result.sort_values(by=['predicted', 'distance'], ascending=True)

from __future__ import division

print 'Accuracy:', len(result[result.actual == result.predicted]) / len(result)

Accuracy: 0.0262645914397

""
Since the recommender recommend issues to developers, this function will determine the
top-n recommendations (issues) to a given developer
""
def get_topn_predictions(developer, n):
    x = sorted_predictions[sorted_predictions.predicted == developer]
    return x.head(n)

# for example, this are the 3 top predictions for the user 'jvalkeal'
get_topn_predictions('jvalkeal', 3)['key']

Out[16]:

<table>
<thead>
<tr>
<th></th>
<th>key</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6033</td>
<td>XD-68</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>XD-3748</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>XD-3747</td>
<td></td>
</tr>
</tbody>
</table>

Name: key, dtype: object
Task 2.1 (2 points)

1. Choose a k-fold strategy or a holdout strategy. Using that strategy, learn the best parameters for the recommender system and evaluate the model as follows:

2. Set N in order to propose a list of N items that a user is expected to find interesting (top-N recommendation).

3. Calculate the PR-curve of the model for different values of N. To build a PR-curve, you will have to:
   - Calculate the Recall of the system
     \[
     \text{Recall} = \frac{\text{liked and shown}}{\text{liked}}
     \]
   - Calculate the Precision of the system
     \[
     \text{Precision} = \frac{\text{liked and shown}}{\text{shown}}
     \]
In [29]:

## Get the top-N predictions and calculate the average precision

MAXN = 50
av_ps = []
av_rs = []

for n in range(1, MAXN):
    ps = []
    rs = []
    # for each developer
    for d in devs:
        # calculate the number of liked (issues that have been done by the developer in the
        liked = set(actual[actual.assignee == d]['key'])
        # calculate the number of shown issues (issues recommended by the system)
        shown = set( get_topn_predictions(d, n)['key'] )

        # calculate the intersection
        liked_shown = list(liked & shown)

        # calculate the precision and recall
        p = len(liked_shown) / len(liked)
        r = 0 if len(shown) == 0 else len(liked_shown) / len(shown)

        # append the results to a list
        ps.append(p)
        rs.append(r)

    av_ps.append(np.average(ps))
    av_rs.append(np.average(rs))

In [30]:

pr = pd.DataFrame({'n': range(0, len(av_ps)), 'recall': av_rs, 'precision': av_ps})

# sort the values by recall
pr2 = pr.sort_values('recall', ascending=True)

In [31]:

pr2.head()

Out[31]:

<table>
<thead>
<tr>
<th>n</th>
<th>precision</th>
<th>recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>2</td>
<td>0.000095</td>
<td>0.010753</td>
</tr>
<tr>
<td>1</td>
<td>0.000095</td>
<td>0.016129</td>
</tr>
<tr>
<td>48</td>
<td>0.011639</td>
<td>0.016623</td>
</tr>
<tr>
<td>47</td>
<td>0.011639</td>
<td>0.016897</td>
</tr>
</tbody>
</table>
In [32]:

```
import matplotlib.pyplot as plt
%matplotlib inline

precision = pr2.precision
recall = pr2.recall

# take a running maximum over the reversed vector of precision values, reverse the
# result to match the order of the recall vector
decreasing_max_precision = np.maximum.accumulate(precision[::-1])[::-1]

fig, ax = plt.subplots(1, 1)
ax.hold(True)
ax.plot(recall, precision, '--b')
ax.step(recall, decreasing_max_precision, '-r')
ax.set_title("PRcurve for recommender system.")
ax.set_xlabel("recall")
ax.set_ylabel("precision")
```

Out[32]:
<matplotlib.text.Text at 0xcc667f0>

In [ ]:

**Task 2.2 (3 points)**

Perform a sanity check of the recommender system you have created.

1. Compare the results in terms of PR-curve of the recommender with the results of a Random recommender.
2. Compare the results in terms of PR-curve of the recommender with a recommender using $k=3$.
3. According to your results, please answer the following question: **How can you improve the performance of your recommender system?**
import random

issues.reset_index(inplace=True)

def get_topn_random_predictions(n):
    actual = issues[['assignee', 'key']]  
    
    rs = []
    for i in range(0,n):
        rs.append(random.choice(issues.index))

    return rs

preds = get_topn_random_predictions(5)
print preds
print issues.iloc[preds]['key']

[525, 239, 2049, 2400, 1543]
525   XD-2759
239   XD-3263
2049  XD-574
2400  XD-83
1543  XD-1247
Name: key, dtype: object
from __future__ import division
import numpy as np

issues = issues[~pd.isna(issues['assignee'])]

av_ps = []
av_rs = []
for n in range(1, 30):
    ps = []
    rs = []
    # for each developer
    for d in issues['assignee'].unique():
        # print d
        # calculate the number of liked (issues that have been done by the developer in the
        liked = set(issues[issues.assignee == d]['key'])
        # calculate the number of shown issues (issues recommended by the system)
        preds = get_topn_random_predictions(n)
        shown = set(issues.iloc[preds]['key'])

        # calculate the intersection
        liked_shown = list(liked & shown)

        # calculate the precision and recall
        p = len(liked_shown) / len(liked)
        r = 0 if len(shown) == 0 else len(liked_shown) / len(shown)

        # append the results to a list
        ps.append(p)
        rs.append(r)

    av_ps.append(np.average(ps))
    av_rs.append(np.average(rs))

pr = pd.DataFrame({'n': range(0, len(av_ps)), 'recall': av_rs, 'precision': av_ps})
pr2 = pr.sort_values('recall', ascending=True)
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 use another language, you must submit the code/scripts used. In this case, each piece of code must clearly indicate by a comment which tasks is addressed.

Additional links

- Python tutorial - [https://docs.python.org/2/tutorial/](https://docs.python.org/2/tutorial/)
- Jupyter installation guide for IPython Notebooks - [https://jupyter.org/install](https://jupyter.org/install)
References


Copyright (c) 2018, Ezequiel Scott All rights reserved.