example_2

July 19, 2019

1 Example 2 - Combining Machine Learning and Operations Research methods to advance the Project Management Practice

In [1]:
```
import json
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

from nltk.stem import LancasterStemmer
stemmer = LancasterStemmer()

from lime.lime_text import LimeTextExplainer

from ortools.graph import pywrapgraph

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to /home/nkanak/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/nkanak/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [2]:
```
# Data has been retrieved from the public accessible Jira instance of the open source p
# Check this URL: https://issues.apache.org/jira .
```

```
         with open('data/hadoop_issues.json') as f:
             issues = json.load(f)
         issues = issues['issues']
```

```
In [3]: extra_stopwords = ["a", "about", "above", "after", "again", "against", "ain", "all", "a
```

```
In [4]: # Keep only the issues with assignee.
        issues = [issue for issue in issues if issue['fields'].get('assignee') is not None]
```

```
In [5]: len(issues)
```

```
Out[5]: 677
```

```
In [6]: # Keep only the issues of the 4 most important employees, i.e. employees with the high
        issues = [issue for issue in issues if issue['fields']['assignee']['key'] in ['stevel@a
```

```
In [7]: fake_names = {
            'stevel@apache.org': 'john',
            'gabor.bota': 'jane',
            'danielzhou': 'johnny',
            'ajisakaa': 'richard'
        }
        for issue in issues:
            issue['fields']['assignee']['key'] = fake_names[issue['fields']['assignee']['key']]
```

```
In [8]: len(issues)
```

```
Out[8]: 174
```

```
In [9]: unique_assignees_to_number_mapping = {assignee: key for key, assignee in enumerate(list
        unique_person_names = sorted([key for key in unique_assignees_to_number_mapping], key=l
        print(unique_assignees_to_number_mapping)
```

```
{'john': 0, 'jane': 1, 'johnny': 2, 'richard': 3}
```

```
In [10]: columns = {
             'class': [issue['fields']['assignee']['key'] for issue in issues],
             'text': [(issue['fields']['description'] if issue['fields']['description'] is not
         }
```

```
In [11]: # Compose and clean up text.
         # The text of each issue is composed of two attrributes, namely description and summa
         # Removal of english stop words is performed as well as lemmatization and stemming.
         # Lemmatization (a Text Normalization technique) is the process of grouping together
         # Stemming is the process of reducing inflected (or sometimes derived) words to their
         # Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that
         # Stemming is different to Lemmatization in the approach it uses to produce root form
         # Also lemmatization and stemming techniques decrease the number of features of each
         for i in range(len(columns['text'])):
             columns['text'][i] = ' '.join([stemmer.stem(wordnet_lemmatizer.lemmatize(word.lowe
```

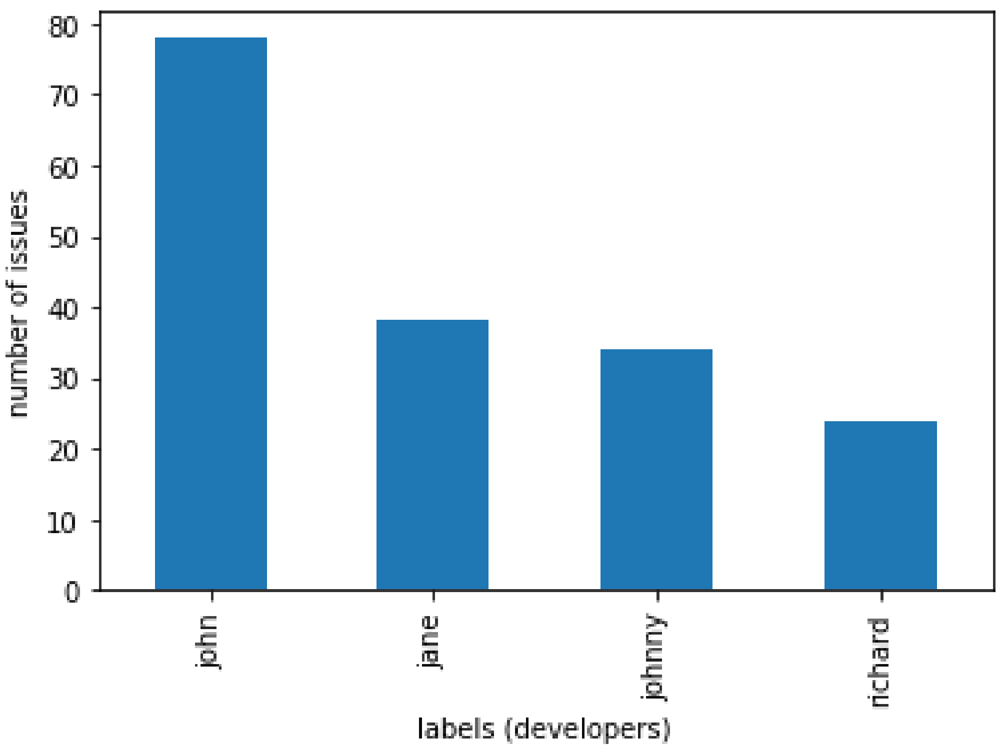
In [12]:
```python
issues_df = pd.DataFrame.from_dict(columns)
#issues_df
```

In [13]:
```python
issues_df['class'].value_counts()
```

Out[13]:
```
john       78
jane       38
johnny     34
richard    24
Name: class, dtype: int64
```

In [14]:
```python
ax = issues_df['class'].value_counts().plot(kind='bar')
ax.set_xlabel('labels (developers)')
ax.set_ylabel('number of issues')
```

Out[14]:
```
Text(0, 0.5, 'number of issues')
```



1.1 Example

In [15]:
```python
#issues_df
```

In [16]:
```python
issues_df['label'] = issues_df['class'].map(unique_assignees_to_number_mapping)
```

```
In [17]: X = issues_df['text']
         y = issues_df['label']
         # Tokenization, tag/feature generation and removal of frequently occured tags/words.
         cv = CountVectorizer(stop_words='english', max_df=0.6)

         X_transformed = cv.fit_transform(X)
         X_train, X_test, y_train, y_test, indices_train, indices_test = train_test_split(X_tra
         print('Train size %s' % X_train.shape[0])
         print('Test size %s' % X_test.shape[0])
         print('Number of features %s' % X_test.shape[1])
```

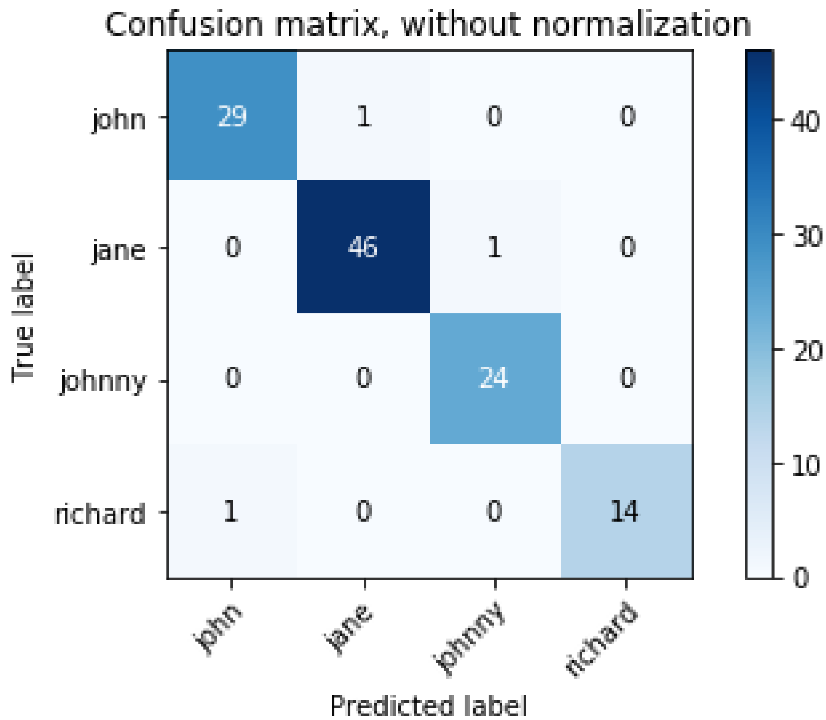
```
Train size 116
Test size 58
Number of features 2403
```

```
In [18]: # Naive Bayes classifier for multinomial models.
         # The multinomial Naive Bayes classifier is suitable for classification with discrete
         # The multinomial distribution normally requires integer feature counts. However, in f
         naive_clf = MultinomialNB()
         naive_clf.fit(X_train,y_train)
         naive_clf.score(X_test,y_test)
```

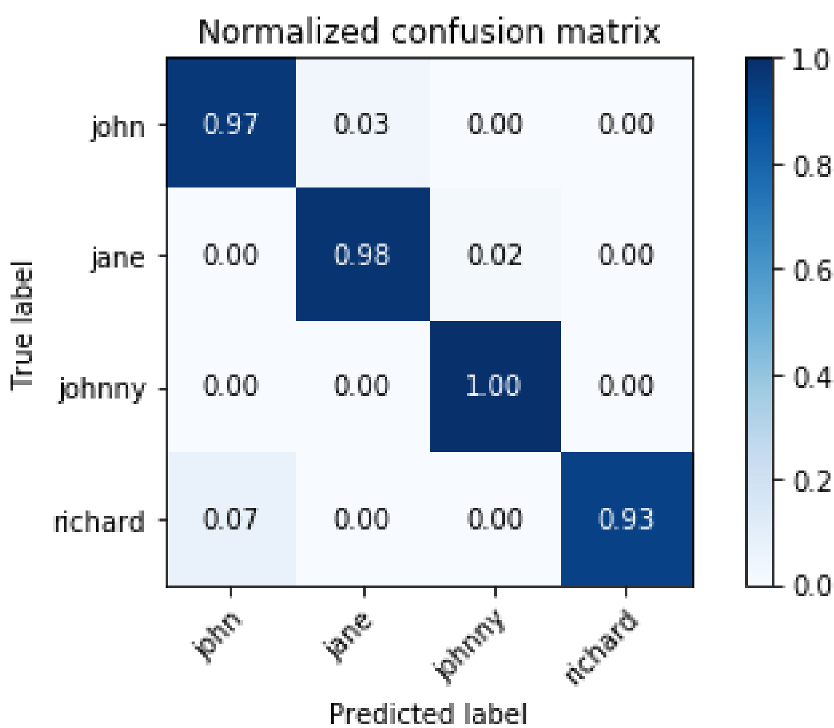
```
Out[18]: 0.8275862068965517
```

```
In [19]: y_predicted = [unique_person_names[i] for i in  naive_clf.predict(X_transformed[indice
         y_true = issues_df.iloc[indices_train]['class'].to_list()
         class_names = unique_person_names

         def plot_confusion_matrix(y_true, y_pred, classes,
                                   normalize=False,
                                   title=None,
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if not title:
                 if normalize:
                     title = 'Normalized confusion matrix'
                 else:
                     title = 'Confusion matrix, without normalization'

             # Compute confusion matrix
             cm = confusion_matrix(y_true, y_pred)
             # Only use the labels that appear in the data
             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

```
    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_true, y_predicted, classes=class_names,
                      title='Confusion matrix, without normalization')
# Plot normalized confusion matrix
plot_confusion_matrix(y_true, y_predicted, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
plt.show()
```



Confusion matrix, without normalization
True label
john
jane
johnny
richard
29
1
0
0
0
46
1
0
0
0
24
0
1
0
0
14
Predicted label
40
30
20
10
0



Normalized confusion matrix
True label
john
jane
johnny
richard
0.97
0.03
0.00
0.00
0.00
0.98
0.02
0.00
0.00
0.00
1.00
0.00
0.07
0.00
0.00
0.93
Predicted label
1.0
0.8
0.6
0.4
0.2
0.0

```
In [20]: # Text Explainer for explaining the selected examples.
         # Reference: https://arxiv.org/abs/1602.04938
         # The Explanations help us to check the reliability and validity of the trained machi
         # The Explanations confirm that the model chooses the right label/class for the right
         explainer = LimeTextExplainer(class_names=unique_person_names)

In [21]: def explain_classification(text, classifier):
             return explainer.explain_instance(text, lambda x: classifier.predict_proba(cv.tran

In [22]: unique_assignees_to_number_mapping

Out[22]: {'john': 0, 'jane': 1, 'johnny': 2, 'richard': 3}

In [23]: def analyze_selected_examples(index):
             print(index)
             print('Real selected label: %s' % issues_df.iloc[index]['class'])
             print('Probabilities of each label: %s' % naive_clf.predict_proba(cv.transform([is
             print('Summary: %s' % issues[index]['fields']['summary'])
             print('Description: %s' % issues[index]['fields']['description'] if issues[index]
             exp = explain_classification(issues_df.iloc[index]['text'], naive_clf)
             exp.show_in_notebook()
             exp.save_to_file('example_explanations/%s.html' % (index))
             return exp

In [24]: analyze_selected_examples(138)

138
Real selected label: john
Probabilities of each label: [[9.63e-01 7.62e-11 4.10e-08 3.70e-02]]
Summary: branch-2 site not building after ADL troubleshooting doc added
Description: Toc error on the ADL troubleshooting doc from HADOOP-15090
{code}
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-site-plugin:3.5:site (default-cli
{code}


<IPython.core.display.HTML object>


Out[24]: <lime.explanation.Explanation at 0x7f77c65a76a0>

In [25]: analyze_selected_examples(65)

65
Real selected label: john
Probabilities of each label: [[9.49e-01 1.43e-02 3.67e-02 8.99e-08]]
Summary: S3 listing inconsistency can raise NPE in globber
Description: FileSystem Globber does a listStatus(path) and then, if only one element is retur
```

```
On S3, if the path has had entries deleted, the listing may include files which are no longer t

While its wrong to glob against S3 when its being inconsistent, we should at least fail gracefu

Proposed
# log all IOEs raised in Globber.getFileStatus @ debug
# catch FNFEs and downgrade to warn
# continue

The alternative would be fail fast on FNFE, but that's more traumatic



<IPython.core.display.HTML object>


Out[25]: <lime.explanation.Explanation at 0x7f77c1adea58>
```

```
In [26]: analyze_selected_examples(85)
```

```
85
Real selected label: johnny
Probabilities of each label: [[8.42e-02 2.02e-04 9.16e-01 3.79e-07]]
Summary: ABFS: Code changes for bug fix and new tests
Description: - add bug fixes.
- remove unnecessary dependencies.
- add new tests for code changes.


<IPython.core.display.HTML object>


Out[26]: <lime.explanation.Explanation at 0x7f77c6db4d68>
```

```
In [27]: analyze_selected_examples(109)
```

```
109
Real selected label: john
Probabilities of each label: [[0.53 0.43 0.01 0.03]]
Summary: Release Hadoop 2.7.7
Description: Time to get a new Hadoop 2.7.x out the door.


<IPython.core.display.HTML object>


Out[27]: <lime.explanation.Explanation at 0x7f77c1a876d8>
```

In [28]:
```
# OR part of the example.
# Target?: To maximize the chance of success of the software company.
# How?: By assigning the employees in such a way that the total relevance is maximize
# Relevance = skills required by an issue vs skills possessed by an employee.
# In our example the 'relevance' is equivalent to the probability (calculated by our
# OR algorithm?: Linear Assignment Problem (LAP) (https://developers.google.com/optim
```

In [29]:
```
selected_example_indeces = [138, 65, 85, 109]
```

In [30]:
```
# Row == employee.
# Column == issue.
# Transpose a list code: list(map(list, zip(*l)))
relevance_of_each_employee_per_issue = list(map(list, zip(*[
    list(naive_clf.predict_proba(cv.transform([issues_df.iloc[i]['text']]))[0]) for i
])))
relevance_of_each_employee_per_issue_percentage = [[int(round(c*100)) for c in row] fo
print('Relevance of each employee per issue: %s' % relevance_of_each_employee_per_issu
```

```
Relevance of each employee per issue: [[96, 95, 8, 53], [0, 1, 0, 43], [0, 4, 92, 1], [4, 0, 0
```

In [31]:
```
def assign_employees_to_issues():
  cost = create_data_array()
  rows = len(cost)
  cols = len(cost[0])

  assignment = pywrapgraph.LinearSumAssignment()
  for worker in range(rows):
    for task in range(cols):
      if cost[worker][task]:
        assignment.AddArcWithCost(worker, task, cost[worker][task])
  solve_status = assignment.Solve()
  if solve_status == assignment.OPTIMAL:
    total_relevance = 0
    for i in range(0, assignment.NumNodes()):
      relevance = relevance_of_each_employee_per_issue_percentage[i][assignment.RightM
      total_relevance += relevance
      print('Employee %s (index:%s) is assigned to issue %s.  Relevance = %d' % (
            unique_person_names[i],
            i,
            selected_example_indeces[assignment.RightMate(i)],
            relevance))
    print()
    print('Total relevance = ', total_relevance)
  elif solve_status == assignment.INFEASIBLE:
    print('No assignment is possible.')
  elif solve_status == assignment.POSSIBLE_OVERFLOW:
    print('Some input costs are too large and may cause an integer overflow.')
```

```
def create_data_array():
  cost = relevance_of_each_employee_per_issue_percentage
  inverse_cost = [[100 - c for c in row] for row in cost]
  #print(cost)
  #print(inverse_cost)
  return inverse_cost
```

In [32]:
```
# The Outcome of the example.
# Evaluation http://www.hungarianalgorithm.com/solve.php?c=96-95-8-53--0-4-92-1--4-0-
assign_employees_to_issues()
```

```
Employee john (index:0) is assigned to issue 65.  Relevance = 95
Employee jane (index:1) is assigned to issue 109.  Relevance = 43
Employee johnny (index:2) is assigned to issue 85.  Relevance = 92
Employee richard (index:3) is assigned to issue 138.  Relevance = 4

Total relevance =  234
```