example_2

June 24, 2019

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

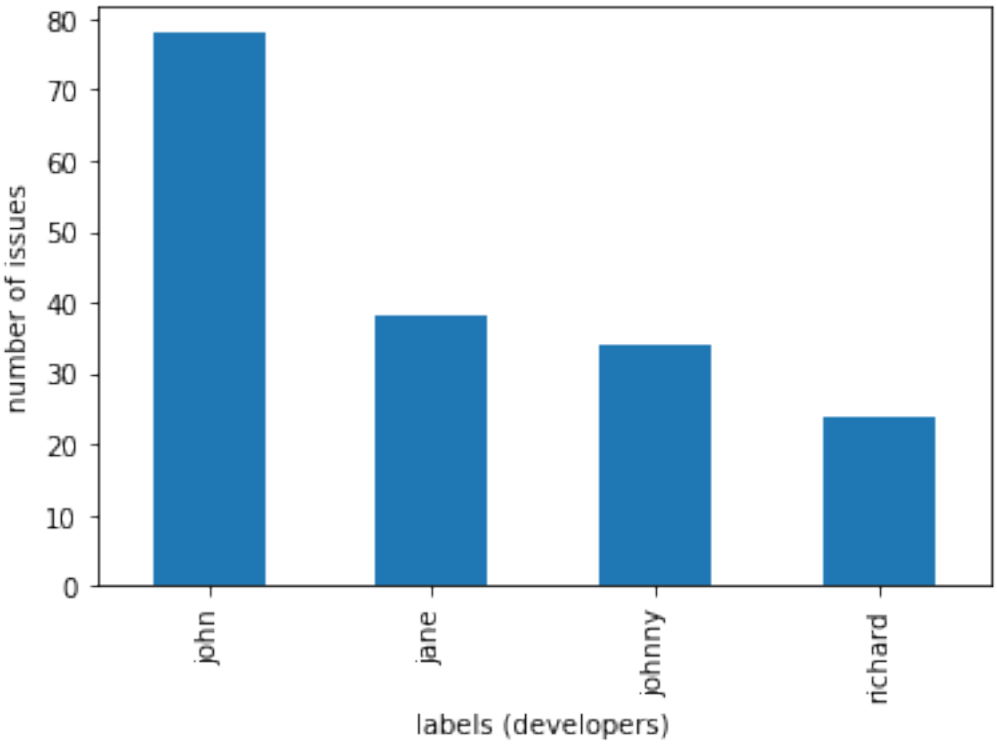
In [1]:
```
import json
import pandas as pd

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

%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)

{'jane': 0, 'johnny': 1, 'john': 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]: issues_df = pd.DataFrame.from_dict(columns)
         #issues_df
```

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

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

In [14]: 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]: #issues_df

In [16]: 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
        naive_clf = MultinomialNB()
        naive_clf.fit(X_train,y_train)
        naive_clf.score(X_test,y_test)
```

Out[18]: 0.8275862068965517

In [19]:
```
        # 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 [20]:
```
        def explain_classification(text, classifier):
            return explainer.explain_instance(text, lambda x: classifier.predict_proba(cv.tra
```

In [21]:
```
unique_assignees_to_number_mapping
```

Out[21]: {'jane': 0, 'johnny': 1, 'john': 2, 'richard': 3}

In [22]:
```
        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([i
            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 [23]:
```
analyze_selected_examples(138)
```

```
138
Real selected label: john
Probabilities of each label: [[7.61860937e-11 4.10151415e-08 9.62965999e-01 3.70339595e-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[23]: <lime.explanation.Explanation at 0x7ff571ffa5c0>

In [24]: analyze_selected_examples(65)

65
Real selected label: john
Probabilities of each label: [[1.43257254e-02 3.67156742e-02 9.48958511e-01 8.99237368e-08]]
Summary: S3 listing inconsistency can raise NPE in globber
Description: FileSystem Globber does a listStatus(path) and then, if only one element is returr

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[24]: <lime.explanation.Explanation at 0x7ff56ff4e8d0>

In [25]: analyze_selected_examples(85)

85
Real selected label: johnny
Probabilities of each label: [[2.02042779e-04 9.15624579e-01 8.41729988e-02 3.79124439e-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[25]: <lime.explanation.Explanation at 0x7ff576893ba8>
```

```
In [26]: analyze_selected_examples(109)
```

```
109
Real selected label: john
Probabilities of each label: [[0.43411654 0.00700768 0.53080359 0.0280722 ]]
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[26]: <lime.explanation.Explanation at 0x7ff56ff54588>
```

```
In [27]: # 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 [28]: selected_example_indeces = [138, 65, 85, 109]
```

```
In [29]: # 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: [[0, 1, 0, 43], [0, 4, 92, 1], [96, 95, 8, 53], [4, 0, 0
```

```
In [30]: 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 [31]:
```
# 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 jane (index:0) is assigned to issue 109.  Relevance = 43
Employee johnny (index:1) is assigned to issue 85.  Relevance = 92
Employee john (index:2) is assigned to issue 65.  Relevance = 95
Employee richard (index:3) is assigned to issue 138.  Relevance = 4

Total relevance =  234
```