

EBC/GNG5125 – Data Science Applications - Summer 2021

**Assignment 1 - Classification**

Submitted by:

|  |  |
| --- | --- |
| Name | Email |
| Michael Khalil | mkhal120@uOttawa.ca |
| Gehad Abo Kamar | gabok033@uOttawa.ca |
| Khadija Taha | khesh072@uOttawa.ca |
| Youssef Metwally | ymetw027@uOttawa.ca |

Submitted to:

Professor: Arya Rahgozar

Teaching Assistance: Migao Wu

**Table of Contents**

**Data Chosen ………………………………………………………………………………...3**

**Preprocessing and Data Cleansing ……………………………………….…………….3**

**Data Partitioning and Labeling ………..…………………………………………………4**

**Feature Engineering…………………………………………………………………….......5**

**Model Building ………………………………………………………………..………….….5**

**Model Evaluation……………………………………………………………………..…..….7**

**Model Selection ……………………………………………………………………..……….8**

**Error Analysis and Visualization ……………………………………………………...….9**

**Analysis of Bias and Variability ………………………………………………………….16**

**Identifying, measuring and control the machine’s thresholds of factors of prediction hardship ……………………………………………………………………..….16**

**Conclusions and Future Work……………………………………………………………16**

**References…………………………………………………….…………………………..…16**

**Data Chosen:**

We chose five books from Gutenberg library of genre of crime.

* Books:
  + Crime and Punishment by Fyodor Dostoevsky
  + Major Barbara by George Bernard Shaw
  + Southern Horrors Lynch Law in All Its Phases by Ida B. Wells-Barnett
  + Financial Crime and Corruption by Sam Vaknin
  + A Love Crime by Paul Bourget

**Preprocessing and Data Cleansing:**

We extracted the author names from the books' text using Regular Expressions.

|  |
| --- |
| author\_Names = []  for book in Books:    author\_Names.append(re.findall(r'Author:\s(.(\w+|.)+)'  , book)[0][0][:-1]) |

Also, we used Regular Expressions to remove garbage characters. We converted text to be all in lower case. In addition to that, we removed stop words and performed stemming to remove additional characters to each word.

|  |
| --- |
| tokenizer = RegexpTokenizer("[\w']+")  for book in Books:    words = re.sub(r'[^\w\s]', '', str(book).lower().strip())    words = tokenizer.tokenize(words)    new\_words = []    for word in words:        if (word not in stop\_words):            new\_words.append(word)    NewWords = []    for i in new\_words:        s = ps.stem(i)        NewWords.append(s) |

**Data Partitioning and Labeling:**

And then, we created random samples of 200 documents of each book, representative of the source input, each of 100 words length and put them in a data frame.

|  |
| --- |
| for x in range( 200):        rand =  random.randint( 5000 , len(NewWords)-100  )        partition = NewWords[ rand : rand+100 ]          part\_200.append( (' '.join(partition),re.findall(r'Author:\s(.(\w+|.)+)', book)[0][0][:-1]) )  df = pd.DataFrame(part\_200 , columns=['Sample', 'Author']) |

**Output:**

Sample Author

0 petrovitch made eager use day capit get rig ex... Fyodor Dostoevsky

1 return call walk door wide open look round wai... Fyodor Dostoevsky

2 game bound win there deni raskolnikov compromi... Fyodor Dostoevsky

3 held hand one hand left slip note pocket saw s... Fyodor Dostoevsky

4 sonia broke hurriedli money dont worri money m... Fyodor Dostoevsky

.. ... ...

995 entir person loverlik appear lent seduct would... Paul Bourget

996 come logic frank momentari separ want ruptur l... Paul Bourget

997 word proport phrase came swept away plan disco... Paul Bourget

998 faster said sever time driver would ever arriv... Paul Bourget

999 impass yawn helen pain way would pain incompre... Paul Bourget

[1000 rows x 2 columns]

**Feature Engineering:**

We used the following transformation methods:

* BOW
* TF-IDF
* n-gram
* LDA
* Word-embedding
* Doc2Vec

**Model Building:**

We splitted the data into 80% training data and 20% testing data.

|  |
| --- |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['Sample'],df['Author'],test\_size=0.2) |

We used, SGD, SVM, Decision Tree, k-Nearest Neighbor, Multi-layer perceptron as classification algorithms for the experimentation.

Then we built a Pipeline to make the code more efficient, organized, and reusable.

|  |
| --- |
| transformations = [                     [('tran', CountVectorizer())],                     [('tran1', CountVectorizer()), ('tran2', TfidfTransformer())],                     [('tran', CountVectorizer(analyzer='word', ngram\_range=(2, 2)))],                     [('tran1', CountVectorizer()), ('tran2', preprocessing.FunctionTransformer(lambda x: x.todense(), accept\_sparse=True)), ('tran3', LinearDiscriminantAnalysis())]                    ]  classifiers = [                 [('clf', SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, random\_state=42, max\_iter=5, tol=None))],                 [('clf', KNeighborsClassifier(n\_neighbors=5))],                 [('clf', DecisionTreeClassifier(random\_state=0))],                 [('clf', svm.SVC())],                 [('clf', MLPClassifier(max\_iter=5))]                ]  models = []  for transform in transformations:    for classifier in classifiers:      text\_clf = Pipeline(transform + classifier)      models.append(deepcopy(text\_clf)) |

We can save the model object as a binary file.

|  |
| --- |
| try:      import dill as pickle  except ImportError:      import pickle  with open('text\_classifier Model Binary file', 'wb') as picklefile:     pickle.dump(text\_clf, picklefile) |

**Model Evaluation:**

We used the ten-fold cross-validation to evaluate the resulted models.

|  |
| --- |
| kf = KFold(n\_splits=10)  for text\_clf in models:    scores = []    for train, test in kf.split(X\_train):      model = text\_clf.fit(X\_train[train], y\_train\_labels[train])      scores.append(metrics.precision\_score(y\_train\_labels[test], model.predict(X\_train[test]), average='micro'))        print("Precision: %0.2f%%" % (np.array(scores).mean()\*100)) |

**Output:**

Precision: 92.63%

-------------------------------------

Precision: 86.75%

-------------------------------------

Precision: 82.88%

-------------------------------------

Precision: 95.00%

-------------------------------------

**Model Selection:**

We selected the X model as the champion model.

We used Grid search parameter optimization as a hyper-parameter optimization algorithm for the champion model.

|  |
| --- |
| parameters = {      # SVM      # 'clf\_\_kernel': ['rbf', 'linear'],      # 'clf\_\_gamma': [1e-3, 1e-4],      # 'clf\_\_C': [1, 10, 100, 1000],      # KNN      # 'clf\_\_n\_neighbors': list(range(1,36))      # DTree      # 'clf\_\_criterion': ['gini', 'entropy'],      # 'clf\_\_max\_depth': [2,4,6,8,10,12]      'tran2\_\_use\_idf': (True, False),      'tran1\_\_ngram\_range': [(1, 1), (1, 2), (2,2)],      'clf\_\_alpha': (1e-2, 1e-3)  }  scorer = metrics.make\_scorer(metrics.precision\_score, average = 'micro')  gs\_clf = GridSearchCV(models[5], parameters, cv=10, n\_jobs=-1, scoring=scorer)  # Fit and evaluate  gs\_clf = gs\_clf.fit(X\_train, y\_train\_labels)  print("Training precision: " + str(gs\_clf.score(X\_train, y\_train\_labels)))  print("Mean cross-validation precision: " + str(gs\_clf.best\_score\_))  print()  print('Best Parameters')  print('----------------')  for param\_name in sorted(parameters.keys()):    print("%s: %r" % (param\_name, gs\_clf.best\_params\_[param\_name]))  print('Report')  print('-------')  print(metrics.classification\_report(y\_train\_labels, gs\_clf.predict(X\_train))) |

**Error Analysis and Visualization:**

First, we plotted the Confusion matrix and FP/FN counts per Author visualizations.

|  |
| --- |
| predicted = gs\_clf.predict(X\_test)  print("Testing precision: " + str(gs\_clf.score(X\_test, y\_test\_labels)))  # estimate bias and variance https://machinelearningmastery.com/calculate-the-bias-variance-trade-off/  \_, bias, var = bias\_variance\_decomp(deepcopy(gs\_clf), X\_train, y\_train\_labels, X\_test, y\_test\_labels, num\_rounds=2)  print('Bias: %.3f' % bias)  print('Variance: %.3f' % var)  cm = metrics.confusion\_matrix(y\_test\_labels, predicted)  metrics.plot\_confusion\_matrix(gs\_clf, X\_test, y\_test\_labels)  # False Negative per Author  group\_by\_true\_label = np.sum(cm, axis=1) - np.diag(cm)  # False Positive per Author  group\_by\_prediction = np.sum(cm, axis=0) - np.diag(cm)  for i in range(len(author\_Names)):    print(str(group\_by\_prediction[i]) + ' texts has been wrongly classified as ' + author\_Names[i])  for i in range(len(author\_Names)):    print(str(group\_by\_true\_label[i]) + ' of ' + author\_Names[i] + ' texts has been wrongly classified')  print(author\_Names)  fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(14, 5))  plt.subplot(1, 2, 1)  plt.bar(range(len(author\_Names)), group\_by\_prediction)  plt.title('False Positive per Author')  plt.xticks(range(len(author\_Names)), author\_Names, rotation=90)  plt.subplot(1, 2, 2)  plt.bar(range(len(author\_Names)), group\_by\_true\_label)  plt.title('False Negative per Author')  plt.xticks(range(len(author\_Names)), author\_Names, rotation=90)  plt.show() |

**Output:**

Testing precision: 0.965

Bias: 0.035

Variance: 0.010

0 texts has been wrongly classified as Fyodor Dostoevsky

0 texts has been wrongly classified as George Bernard Shaw

6 texts has been wrongly classified as Ida B. Wells-Barnett

1 texts has been wrongly classified as Sam Vaknin

0 texts has been wrongly classified as Paul Bourget

2 of Fyodor Dostoevsky texts has been wrongly classified

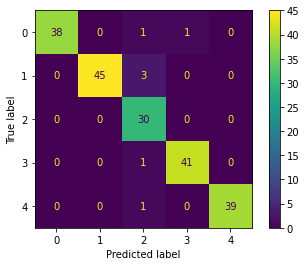
3 of George Bernard Shaw texts has been wrongly classified

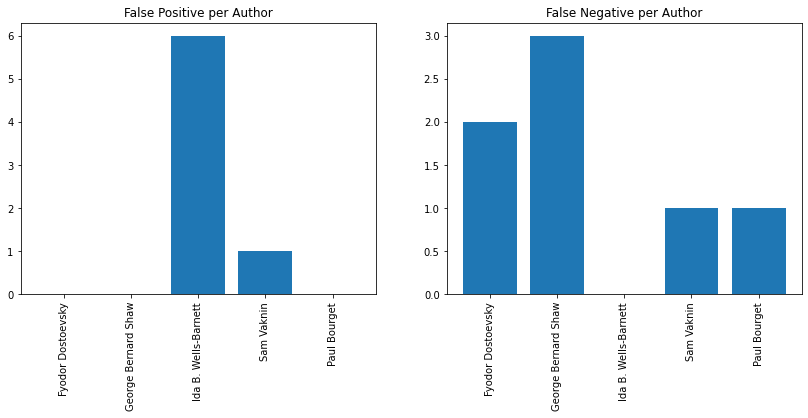
0 of Ida B. Wells-Barnett texts has been wrongly classified

1 of Sam Vaknin texts has been wrongly classified

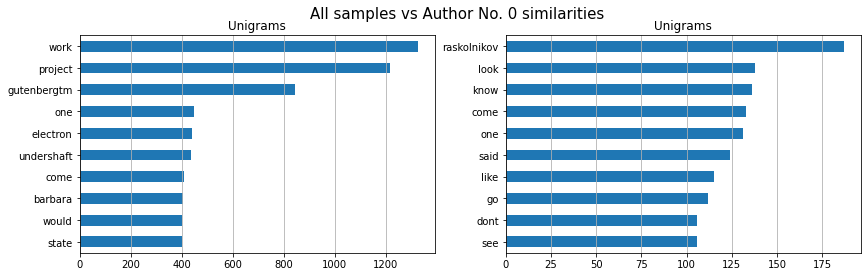
1 of Paul Bourget texts has been wrongly classified

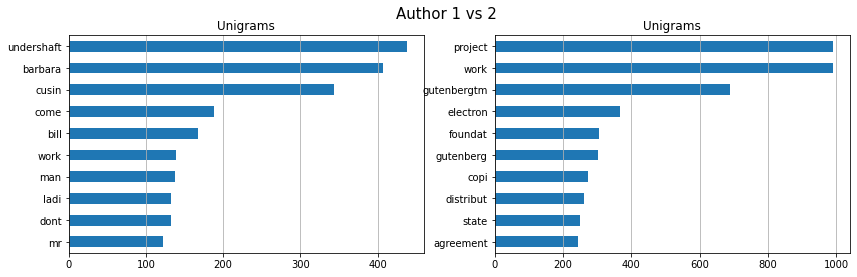
['Fyodor Dostoevsky', 'George Bernard Shaw', 'Ida B. Wells-Barnett', 'Sam Vaknin', 'Paul Bourget']

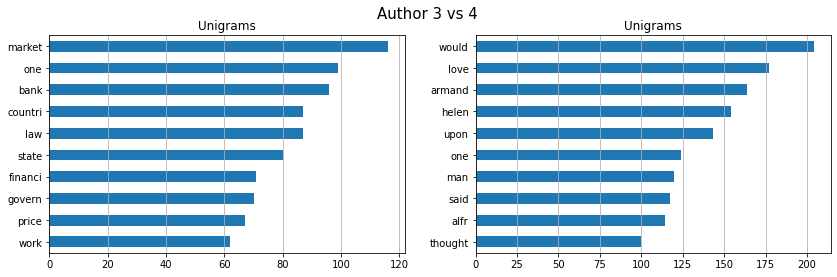




Then we plotted the most frequent words that each author used in his book.







Authors 0 and 2 has no FN, the reason maybe is because they have words that are not common and unique to them in their lists of the most frequent word. For example, author 0 has 'raskolnikov' and author 2 has 'afroamerican' and 'gutenbergtm'.

That could mean that the model is biased towards authors that use unique words.

|  |
| --- |
| fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(14, 4))  fig.suptitle("All samples vs Author No. 0 similarities", fontsize=15)  top=10  lst\_tokens = nltk.tokenize.word\_tokenize(' '.join(np.array(df['Sample'])))  dic\_words\_freq = nltk.FreqDist(lst\_tokens)  dtf\_uni = pd.DataFrame(dic\_words\_freq.most\_common(),                         columns=["Word","Freq"])  dtf\_uni.set\_index("Word").iloc[:top,:].sort\_values(by="Freq").plot(                    kind="barh", title="Unigrams", ax=ax[0],                    legend=False).grid(axis='x')  ax[0].set(ylabel=None) |

Let's see the samples that threw off the model for further analysis.

|  |
| --- |
| for input, prediction, label in zip(X\_test, predicted, y\_test\_labels):    if prediction != label:      print(input[:70] + '... has been classified as ', prediction, 'and should be ', label)      fig, ax = plt.subplots()      fig.suptitle("Classified as " + str(prediction) + " and should be " + str(label), fontsize=15)      lst\_tokens = nltk.tokenize.word\_tokenize(input)      dic\_words\_freq = nltk.FreqDist(lst\_tokens)      dtf\_uni = pd.DataFrame(dic\_words\_freq.most\_common(),                            columns=["Word","Freq"])      dtf\_uni.set\_index("Word").iloc[:top,:].sort\_values(by="Freq").plot(                        kind="barh", title="Unigrams", ax=ax,                        legend=False).grid(axis='x')      ax.set(ylabel=None) |

**Output:**

employe expend consider effort identifi copyright research transcrib p... has been classified as 2 and should be 1

full extent permit us feder law state law foundat princip offic locat ... has been classified as 2 and should be 0

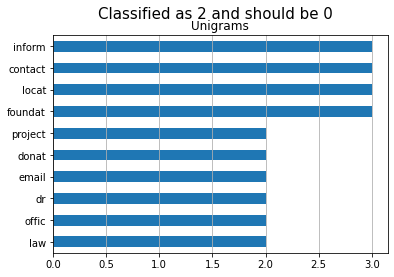
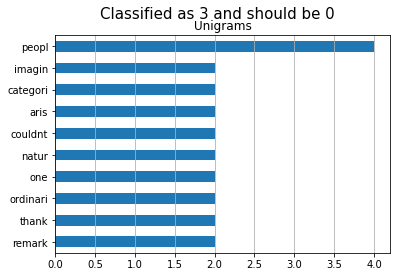
remark witti thank tell distinguish extraordinari peopl ordinari one s... has been classified as 3 and should be 0

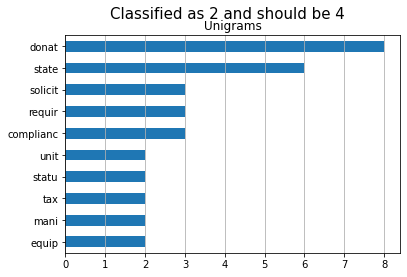
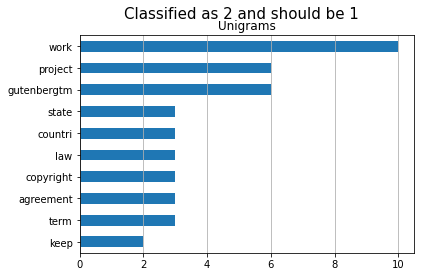
gutenbergtm work complianc term agreement keep project gutenbergtm nam... has been classified as 2 and should be 1

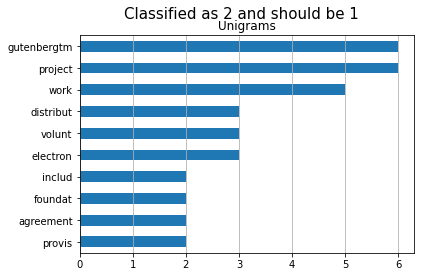
licens share without charg other particular work one individu work pro... has been classified as 2 and should be 3

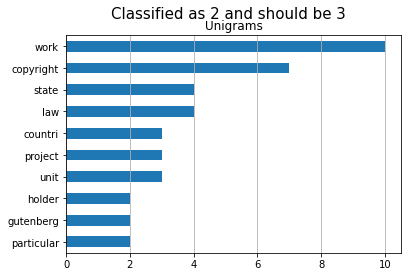
state law invalid unenforc provis agreement shall void remain provis 1... has been classified as 2 and should be 1

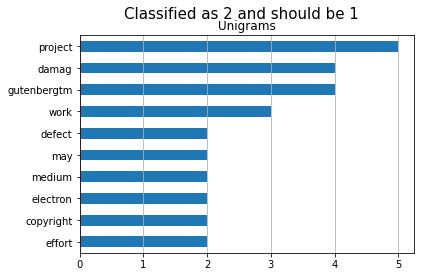
licens work freeli distribut machineread form access widest array equi... has been classified as 2 and should be 4











We can see that False Positives of Author 2 have some frequent words that are the same of the top frequent words for Author 2. Also they use words like 'gutenberg' which is a little unique for Author 2

**Treating the errors**

After examining the wrong predictions, we have found out that there is an intersection between books at the beginning and end of each book.

As they all have been dowloaded from the ‘Gutenberg’ corpus.

So, we altered the code to skip these parts when generating samples. So, let’s perform some error analysis of the results.

First, the Confusion matrix and FP/FN visualization.

**Output:**

Testing precision: 0.965

Bias: 0.005

Variance: 0.005

0 texts has been wrongly classified as Fyodor Dostoevsky

0 texts has been wrongly classified as George Bernard Shaw

0 texts has been wrongly classified as Ida B. Wells-Barnett

0 texts has been wrongly classified as Sam Vaknin

0 texts has been wrongly classified as Paul Bourget

0 of Fyodor Dostoevsky texts has been wrongly classified

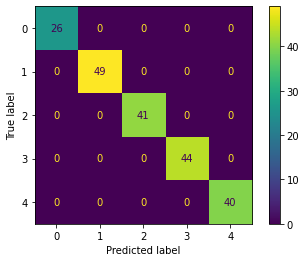
0 of George Bernard Shaw texts has been wrongly classified

0 of Ida B. Wells-Barnett texts has been wrongly classified

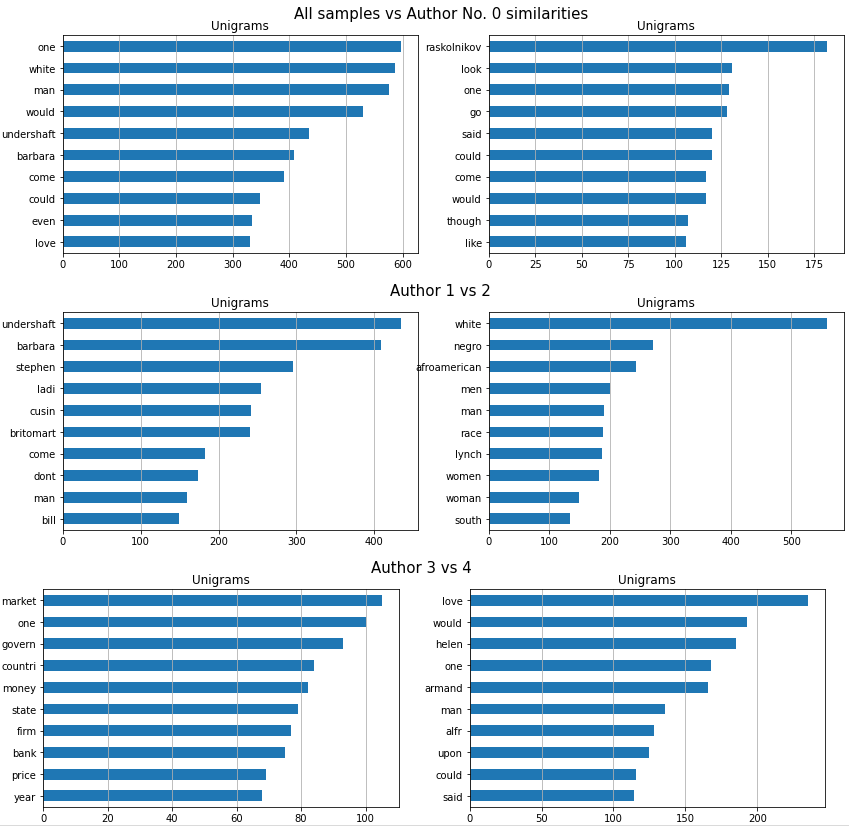
0 of Sam Vaknin texts has been wrongly classified

0 of Paul Bourget texts has been wrongly classified

['Fyodor Dostoevsky', 'George Bernard Shaw', 'Ida B. Wells-Barnett', 'Sam Vaknin', 'Paul Bourget']



Most frequent words



The model shows 100% precision and has no wrong predictions in training and testing.

**Analysis of Bias and Variability:**

The model shows higher bias when there it is more under fitted.

As it was mentioned, there was more bias towards some authors. When the common parts of the books were included, the model was a little more under fitted in that situation.

The model needs to have a good balance between bias and variance so, it can avoid being under fitted or overfitted.

An overfitted model is low on the bias since it can fit all the training data, but it is also high on variance for the same reason. An under the fitted model is high on the bias since it cannot fit all the training data and its predictions are sharp and not sensitive to changes, but it is low on variance for the same reasons. That is called the bias-variance trade-off.

**Identifying, measuring and control the machine’s thresholds of factors of prediction hardship:**

Using the same steps of error analysis, but lowering the number of words per sample to 20, for example.

We obtained the following results:

Training precision: 0.86875

Testing precision: 0.465

Bias: 0.715

Variance: 0.372

55 texts has been wrongly classified as Fyodor Dostoevsky

21 texts has been wrongly classified as George Bernard Shaw

19 texts has been wrongly classified as Ida B. Wells-Barnett

4 texts has been wrongly classified as Sam Vaknin

8 texts has been wrongly classified as Paul Bourget

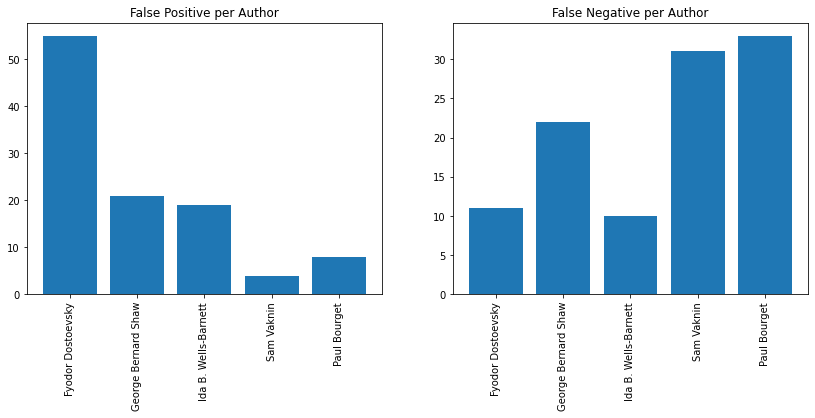
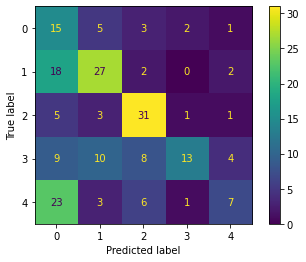
11 of Fyodor Dostoevsky texts has been wrongly classified

22 of George Bernard Shaw texts has been wrongly classified

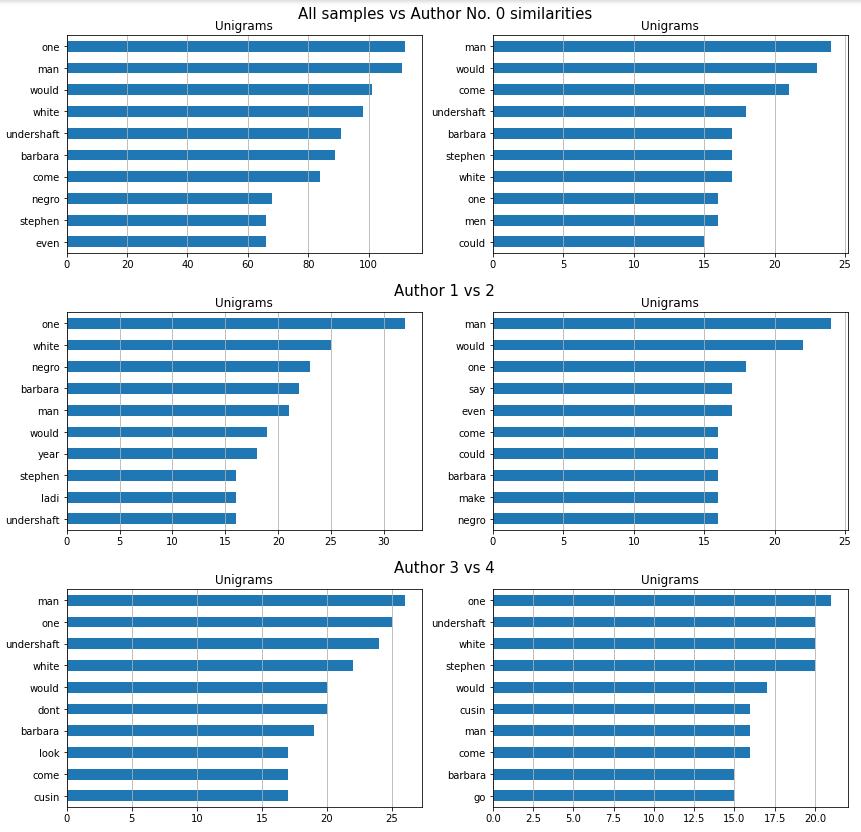
10 of Ida B. Wells-Barnett texts has been wrongly classified

31 of Sam Vaknin texts has been wrongly classified

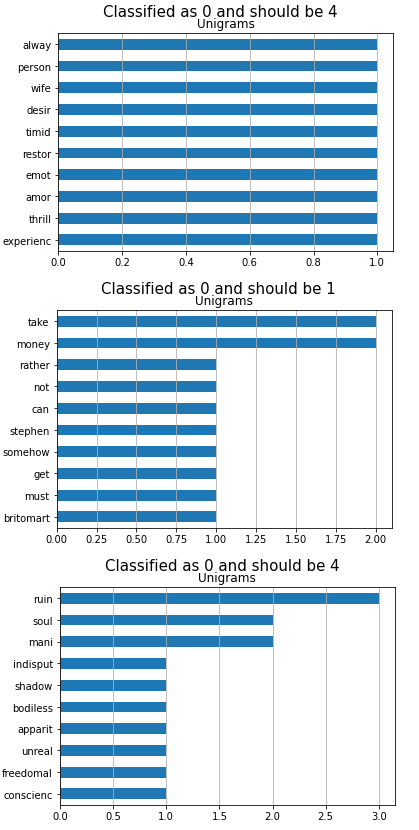
33 of Paul Bourget texts has been wrongly classified



**Most frequent words**



**Examples that threw off the machine**



The most frequent words for each have many common words. We believe that threw off the machine.

The bias and variance are high because the model is under fitted.

There is misclassification of books related to other authors that the model classified to the first author in the data set.

To improve this model, we need to decrease the bias by feeding more data to the model.

**Conclusion and Future Work:**

We need to move forward into two aspects. Regarding the data, we need to widen the range of books used in this study. Due to we have got a 100% per cent of precision in both the training and testing phase.

We also need to add an extra cleaning phase that filters each book from the most frequent words. That can add fairness to the TF-IDF transformer, which has used in this study and other transformers, in addition to a combination of most well performance transformations for accuracy improvement if needed.

**References:**

* scikit-learn documentation: <https://www.scikit-learn.org/>
* Project Gutenberg: <https://www.gutenberg.org/>
* <https://machinelearningmastery.com/calculate-the-bias-variance-trade-off/>
* Lecture notes and notebooks