Author Age and Gender Prediction from Written Samples

Matthew Bowers, Jayam Patel, Jonas Rogers, Qing Xu

CS534: Artificial Intelligence

Table of Contents

Abstract

Author profiling of anonymously written text can be hugely valuable. With increasing commenting and other participation online, often from anonymous accounts, detecting age, gender or anything else about the author can yield useful information. In order to profile the writing, natural language processing was utilized to extract features from the text. The features extracted were word and character based N-grams in this project. The extracted features were sent through multiple different machine learning classification algorithms including Naive Bayes, k-Nearest Neighbors, Maximum Entropy, Support Vector Machines, and Decision Trees. Using this process, author age was able to be classified into 10-year bins with up to 68% accuracy, and detect the gender with up to 65% accuracy. These results are consistent with other work that has been done.

Introduction and Background

We yield a reliable prediction on the range of ages and genders of the author given a paragraph they wrote based on some of the most promising researching areas in AI, Natural Language Processing and Machine Learning, and applied to some of the popular algorithms within. Natural language processing, often abbreviated as NLP, refers to the ability of a computer to translate and interpret the human language as it is spoken or written, and digest it into meaningful information. To some degree, NLP has close cooperation with machine learning, a specific type of AI that analyzes and makes use of patterns in data to improve a program's understanding of a given material. We assume that people in different age groups and genders have distinct patterns when they are writing and those patterns can be identified and then applied by machine learning algorithms. So based on the patterns found, we can detect what age groups and genders the authors belong to.

There have been some useful techniques in text mining already such as content based, POS tagging, sentence length average, vocabulary diversity, stop words and N-grams and there are also some mature classification methods existed in machine learning as well. Some researchers have used linear regression model based on shallow text features for extracting the age of author from text found on blogs, telephone conversations, and online forum posts[1]. Some researchers have also attempted to use natural language processing for not only gender and age but also to predict the personality of the person by using "open-vocabulary technique" where the data itself drives a comprehensive exploration of language that distinguishes people and finding connections between them which normal analysis cannot predict[2]. Within those, Moadh Guefrech(2015) posed a text mining project on gender and age detection applying Naive Bayes Classifier on the extracted age features from an unbalanced social media dataset[3]. And

Peersman, Claudia, Walter Daelemans, and Leona Van Vaerenbergh(2011) sorted the authors into age groups of minus16 and plus25 and achieved about 70% of accuracy applying SVM in the charting data from Netlog[4].

legitimize or delegitimize the writing could be useful and profitable when profiling anonymous writers but direct verification is not available or they can only do the detection through internet and computer without video chats. It also allow marketing companies to know about the people who like or dislike their product based on the reviews

Work Done

Data Selection and Feature Extraction

The data was taken from PAN 2013[6] author profiling test corpus 2. This data included both English writing samples and Spanish writing samples. Only English language samples were used as the volume of data was already significant and this would allow the elimination of known English stop words . There were 25,440 English language samples each in their own .xml file. each file was one side of a chat conversation and each generally contained dozens of sentences.

The sklearn toolkit [5] in python was utilized to extract the feature from this corpus of text. Two different methods of extraction were implemented, Word-based N-grams and Character based N-grams. For the word features both n=1 and n=2 N-grams (mono-grams and bi-grams) were extracted. While for the character features both n=3 and n=4 N-grams were extracted.

Feature Selection and tf-idf Transformation

For the word N-grams the English stop words were filtered out during the extraction process. This utilized the sklearn stop word library which is composed of 318 words. At this point the word gram matrix consisted of 4,024,375 features. Then the number of features in both the word gram set and the character gram set were reduced through the variance threshold approach. Only one threshold was implemented which was chosen as 0.16 a somewhat low variance in order to reduce the data size.

The data was then transformed from this simple count of the number of times a feature was present in a given document to a tf-idf form. This relates the term frequency to the document frequency by multiplying the term frequency by the inverse of the document frequency. this

inverse document frequency is calculated by

$$idf(t) = log\frac{n_d}{1+df(d,t))}$$

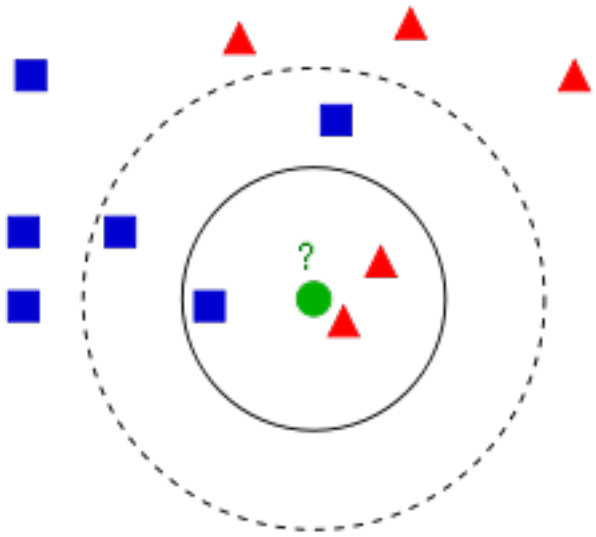
The rows are then normalize to have unit norm using

$$v_{norm} = \frac{v}{||v||^2} = \frac{v}{\sqrt{v_1^2+v_2^2+...+v_n^2}}$$

resulting in the features having a value between 0 and 1.

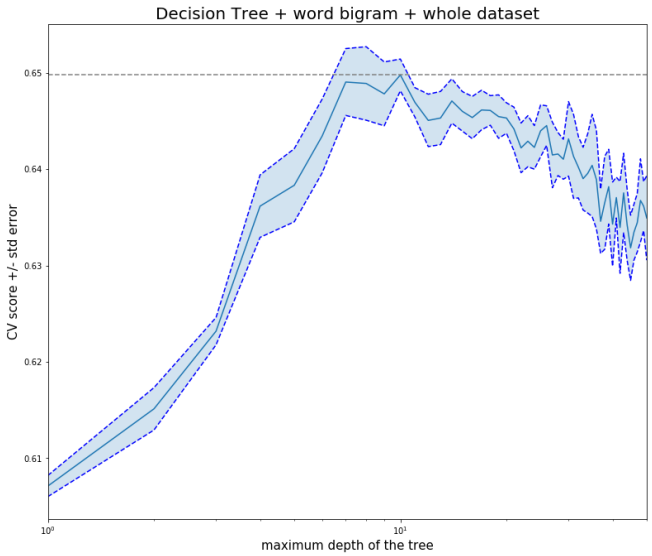
kNN Classifier

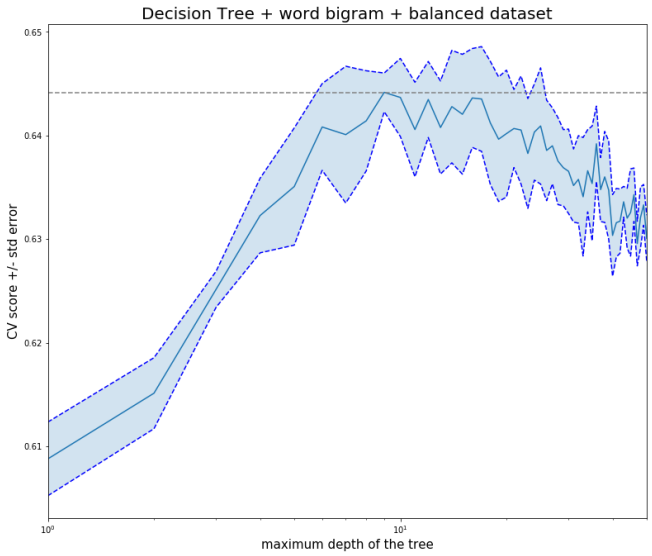
The k-Nearest Neighbors classifier is one of the simplest machine learning algorithms used for classification of data. “k” is a user defined constant used to determine the nearest training samples to the query point and predict the outcome based on the labels of these neighbors. The distance between these points can be found by using Euclidean distance for calculating metric distance and Hamming Distance can be used to calculating distance for text (non-metric) distance.



Decision Trees Classifier

Besides of predicting the age from the authors getting the accuracy of the test data, which is our original goal, we also tried to find a way to improve the performance of the classifiers, so we firstly applied a grid research technique to explore the parameters in decision tree to see how will the accuracy differ with different parameters in the same dataset. Furthermore, since we have an unbalanced dataset which contains 7% of examples in an age range of 10s, 36% of age 20s and 37% of age 30s, if we place an instance randomly, it still have 57% probability to be tossed into the 30s class. This will impact the confidence of the classifier. So we analyze the difference between the different sets using as well. These techniques can also be further used in the choice of KNN and smoother in SVM.





Going through many experiments, we found that the performance of the chatting dataset is most sensitive to the maximum depth of the tree. Here goes the trade-off. If the depth is high

then it means that the tree is very close to the training data than necessary because it means that the classifier takes too many "features" from the training data that is not existed in the testing data. But if the tree depth is low, then it might go over the features that is relatively weaker but still useful. So it makes sense to get a best parameter for a specified kind of data as it improve the performance greatly.

As the figures on the left showed above, the results of the exploration of the maximum depth indicates that the accuracy is firstly increasing from depth 1 to depth 10, and then slowly decreasing after the best one. It make sense to explain that depth below 10 ignore too many useful predictors and depth above 10 lead to overfitting. And on the right figure, we use the same technique on the balanced data and get a result that is pretty similar to the unbalanced one. The balance problem is not impacting the performance of the classifier. But we can see that the the standard error (shown as the blue areas)from the cross validation result is larger than the unbalanced one mostly because the training samples are smaller so the classification is less stable.

Maximum Entropy

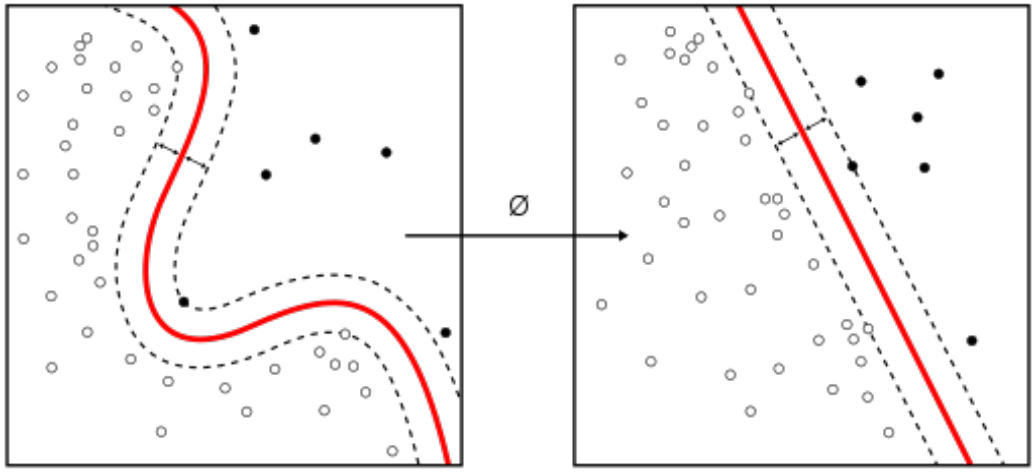
The Maximum Entropy classifier, also known as the Logistic Regression Classifier, is a powerful algorithm. The classifier is used to categorize samples into categorical groupings. The most basic version only differentiates between two categories, but the advanced version used in this project categorizes into multiple categories. Although the Maximum Entropy classifier generally classifies objects into non-ordered bins, the bins used here have an implicit ordering. The classifier does not take this knowledge into account, however, and may have returned in better results if that information were taken into consideration.

The classifier works by fitting a logistic curve to the data as best as possible for each

independent variable or feature. Then after calculating the regression for each feature, it determines the impact each has on the outcome in probabilities. This allows for very quick testing, and indeed the Maximum Entropy classifier was one of the quickest to run, processing and testing 1.5GB of data in under 30 minutes on the test machine. The use of probabilities also allows for good insight into the most impactful features with potential use for combinations with other models in the future.

Support Vector Machines

Support Vector Machines (SVM) is a machine learning technique to analyze data using classification and regression. The discrimination of the features is formally achieved by using separating hyperplanes of high/infinite dimension. The hyperplanes along with the margin between the features which help in converting the characteristics of the data to useful features. For complicated data usually, the features are not linearly separable and hence a much higher-dimension space is used for mapping the data making the separation easier in space, and for this we introduce a kernel function to convert a non-linear separation to a linear separation. Kernel Machines enables the user to work in high-dimension space by computing the inner products between the images of all pairs of data in the feature space. An example of the kernel machine is shown in the figure below:



Kernel Machine change for each application and for the project we tested SVM using following types of Kernel:

Linear Kernel,

$$K(x, y) = x^T.y$$

and, Polynomial Kernel,

$$K(x, y) = (x^T.y + c)^p$$

where, x,y = input vectors

p = power of polynomial

c = constant

Proposed Experiments and Outcomes

Datasets Used

For testing, the corpus of text used was provided by the PAN organization[6]. It included 3 years of conference sample text with author profiling information: age group, gender, and a few others. Age groups were divided by 10 years as such: 10s, 20s, and 30s. They were not divided evenly, but the machine learning classifiers utilized here did not have even groups as a requirement. Additionally, the written samples were of a significant length to ensure that enough words and grammar were included in the samples to give enough information about the writing. Originally, tweets were considered as well, but their grammatical structure is often mangled due to the character limit and prominence of hashtags. The writing samples in the dataset used here were more formal than tweets, but not fully formal published writing, so they are fairly representative of what may be used in a letter or email. The data, once fully collected, contained around 20,000 writing samples, each from a different author.

With feature extraction and selection, the 1 and 2 N-grams with words resulted in just over 2300 features. This was sufficient to yield good results with most classifiers used. The character-based N-grams were extracted with 3 and 4 as the values for N, and resulted in around 25,000 features. Although this seems like it may be more predictive, it yielded fairly similar results, even worse in some cases. The data was partitioned into training and testing sets, with 67% of the data going into the training data and 33% going to the test set. The same partition was used for all classifiers. Results from the classifiers are presented in the following tables.

Classifier's Accuracy

The accuracy of the classifiers tested is presented in the following two tables

Table 1: Word-based Ngram Feature Extraction

Classifier	Age		Gender	
	Training Acc.	Test Accuracy	Training Acc.	Test Accuracy
kNN	0.755	0.658	0.743	0.607
Decision Trees	0.715	0.653	DNF	DNF
Max Entropy	0.710	0.683	0.682	0.648
SVM (Linear Kernel)	0.735	0.680	0.702	0.631
SVM (Poly Kernel)	0.567	0.569	0.500	0.502

Table 2: Character-based Ngram Feature Extraction

Classifier	Age		Gender	
	Training Acc.	Test Accuracy	Training Acc.	Test Accuracy
kNN	DNF	DNF	DNF	DNF
Decision Trees	0.970	0.631	DNF	DNF
Max Entropy	0.756	0.691	0.732	0.666
SVM (Linear Kernel)	0.770	0.532	0.803	0.514
SVM (Poly Kernel)	DNF	DNF	DNF	DNF

*DNF = Did Not Finish (Took too long to run and did not finish computing accuracies)

Conclusion

The results produced by the classifiers is good, but not great. Our best results were better than all of the competitors in PAN 2013 [7]. However some strategies have achieved results in the 90%+ accuracy level [8][9]. The age prediction accuracy obtained using word-based N-grams with the Maximum Entropy and SVM classifiers (68%) is very good, especially considering that a random selection would only be correct 33% of the time. The gender accuracies were similar, but this is less impressive since a random guess would likely yield accuracies around 50%. One plausible reason for this is that gender just isn't as impactful a parameter on writing style as age. It is still good, with 65% accuracy in the best cases, but gender is clearly not as strong a predictor as age is for writing. The maximum entropy classifier performing very well was expected due to the nature of it not assuming independence between features.

As for feature extraction and selection, this was all done based on the N-grams, but it is clear that other features may be extremely predictive as well, such as sentence length and word length. It is expected that the addition of these features would have improved the accuracies of all the algorithms as it is expected that sentence length and average word length would be highly predictive. Some previous research has used these features to great success in predicting age. Another feature that may have been useful is the size of an author's vocabulary, namely the number of different words used. This may have been especially predictive for age, and somewhat predictive for gender.

Future Work

Following are some suggestions for methods to increase the accuracies of the detections

in any future work done on this project:

First is the extraction of additional types of features such as POS usage, words per sentence, size of vocabulary, average number of syllables, and word length. This information would be valuable in feeding to the classifiers to produce better results. These features are all highly indicative of language development level and would be very useful in determining young authors.

Tuning the current feature selection method and implementing additional methods would reduce the amount of information to use in classification by getting rid of features that do not improve the prediction algorithms. These additional selection methods include univariate feature selection, tree based feature selection, and recursive feature selection. The results from the classifiers could also help inform which features are being used more than others. Additional classifier parameters could also be tuned to improve the results.

References

[1] Nguyen, Dong, Noah A. Smith, and Carolyn P. Rosé. "Author age prediction from text using linear regression." Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities. Association for Computational Linguistics, 2011.

[2] Schwartz, H. Andrew, et al. "Personality, gender, and age in the language of social media: The open-vocabulary approach." PloS one 8.9 (2013): e73791.

[3] http://pars.ie/publications/teaching/text_mining/arch_reports/ss15_mg_report.pdf

[4] Peersman C, Daelemans W, Van Vaerenbergh L. Predicting age and gender in online social networks[C]//Proceedings of the 3rd international workshop on Search and mining user-generated contents. ACM, 2011: 37-44.

[5] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825-2830.

[6]http://www.uni-weimar.de/medien/webis/corpora/corpus-pan-labs-09-today/pan-13/pan13-data/pan13-author-profiling-test-corpus2-2013-04-29.zip

[7] Rangel, F., Rosso, P., Koppel, M. M., Stamatatos, E., & Inches, G. (2013). Overview of the author profiling task at PAN 2013. In *CLEF Conference on Multilingual and Multimodal Information Access Evaluation* (pp. 352-365). CELCT.

[8] Tam, J., and Martell, C. 2009. Age Detection in Chat. In Proceedings of the 3rd IEEE International Conference on Semantic Computing. (Berkeley, USA, September 14-16, 2009). DOI=10.1109/ICSC.2009.37.

[9] Pentel, A. (2015). Automatic Age Detection Using Text Readability Features. In

EDM (Workshops).

Appendix A: Data Extraction Implementation

```
# Feature Extraction and Selection Code
```

```
# Import the Extractor from sklearn
from sklearn.feature_extraction.text import CountVectorizer

# Import the numpy for writing to csv
import numpy as np

# Load the Corpus (as a list of xml files) Download corpus 2
#http://www.uni-weimar.de/medien/webis/corpora/corpus-pan-labs-09-today/pan-13/pan13-
data/pan13-author-profiling-test-corpus2-2013-04-29.zip
corpus = open('Files13.csv').read().splitlines()

# Get the vectorizer setup to (read my input file as a list of files, set it to words, set it
to mono grams and bi grams, utilize english stop words)
vectorizer = CountVectorizer(input='filename', analyzer='word', ngram_range=(1,2),
stop_words='english')

# Extract word features
X1 = vectorizer.fit_transform(corpus)

# Feature selection
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
Y = sel.fit_transform(X1)

# Create tfidf
from sklearn.feature_extraction.text import TfidfTransformer
transformer = TfidfTransformer(smooth_idf=False)
tfidfY = transformer.fit_transform(Y)

# Create Excel file for tfidf of words in array form
P1 = tfidfY.toarray()

# Because some reason the xlsxwriter code tranposes the data I tranpose first
P1=P1.transpose()
import xlsxwriter

workbook = xlsxwriter.Workbook('arraywordtfidf.xlsx')
worksheet = workbook.add_worksheet()

row = 0

for col, data in enumerate(P1):
    worksheet.write_column(row, col, data)

workbook.close()

# Create Excel file for Bag of Words in array form
P2 = Y.toarray()

# Because some reason the xlsxwriter code tranposes the data I tranpose first
P2=P2.transpose()

workbook = xlsxwriter.Workbook('arrayword.xlsx')
worksheet = workbook.add_worksheet()

row = 0

for col, data in enumerate(P2):
    worksheet.write_column(row, col, data)
```

```
workbook.close()

# Do it over again but for characters instead of words

# Get the vectorizer setup to (read my input file as a list of files, set it to characters,
set it to tri grams and quad grams)
vectorizerB = CountVectorizer(input='filename', analyzer='char', ngram_range=(3,4))

# Extract character features
X2 = vectorizerB.fit_transform(corpus)

# Feature selection
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
Z = sel.fit_transform(X2)

# Create tfidf
transformerB = TfidfTransformer(smooth_idf=False)
tfidfZ = transformerB.fit_transform(Z)

# Create csv file for tfidf of characters in array form
P3 = tfidfZ.toarray()

# Could not fit into Excel file, plus this is probably better anyways
np.savetxt('arraychartfidf.csv', P3, fmt='%g', delimiter=',', newline=';')

# Create csv file for Bag of words(characters) in array form
P4 = Z.toarray()
np.savetxt('arraychar.csv', P4, fmt='%g', delimiter=',', newline=';')
```

Appendix B: Decision Tree Implementation

```
import pandas as pd
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import math
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.grid_search import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
import numpy as np
from sklearn.naive_bayes import MultinomialNB

# read the labels and bigram words frequency
labels=pd.read_excel("categorizing info.xlsx",header=None)
labels.columns=["filename","gender","age","others"]
age=labels["age"].as_matrix()
gender=labels["gender"].as_matrix()
print(age[:5])
print(gender[:5])
balance_ratio=math.floor(0.07*len(labels))
balance_ratio
# the tdidf words features is for decision tree
bigram_word_tdidf=pd.read_csv("arraywordtfidf.csv",header=None)
len(bigram_word_tdidf)
#len(bigram_char_tdidf)
len(age)
len(gender)
dt=DecisionTreeClassifier()
param_grid={'max_depth':range(1,51)}


#split the train test data
word_train, word_test, age_train, age_test = train_test_split(bigram_word_tdidf, age,
test_size=0.2, random_state=42)
# testing different depth on age
rsearch = RandomizedSearchCV(estimator=dt, param_distributions=param_grid,n_iter=50)
rsearch.fit(word_train,age_train)
# summarize the results of the random parameter search
print("dt word whole best score on age: ",rsearch.best_score_)
print("dt word whole best depth on age: ",rsearch.best_estimator_.max_depth)
dt=DecisionTreeClassifier(max_depth=rsearch.best_estimator_.max_depth)
dt.fit(word_test,age_test)
print("dt word whole test score on age: ",dt.score(word_test,age_test))


# In[43]:


def plotAccuracy_dt(rs,xlabel,title):
    # get the information from the grid research
    depths=list()
    scores = list()
    scores_std = list()
    for e in rs.grid_scores_:
        depths.append(e.parameters.get("max_depth"))
        scores.append(e.mean_validation_score)
        scores_std.append(np.std(e.cv_validation_scores))
```

```
    # plot out the mean accuracy and standard errors from cross validation of different alphas

    plt.figure().set_size_inches(13, 11)
    plt.semilogx(depths, scores)

    # plot error lines showing +/- std. errors of the scores
    std_error = scores_std / np.sqrt(3)

    plt.semilogx(depths, scores + std_error, 'b--')
    plt.semilogx(depths, scores - std_error, 'b--')

    # alpha=0.2 controls the translucency of the fill color
    plt.fill_between(depths, scores + std_error, scores - std_error, alpha=0.2)
    plt.ylabel('CV score +/- std error',fontsize=15)
    plt.xlabel(xlabel,fontsize=15)
    plt.title(title,fontsize=20)
    plt.axhline(np.max(scores), linestyle='--', color='.5')
    plt.xlim([depths[0], depths[-1]])
    #plt.ylim([ np.min(scores), np.max(scores)])
    #plt.vlines(rs.best_estimator_.max_depth,ymin=linestyle='dashed')
    fn=title+".png"
    plt.savefig(fn)


# In[44]:

plotAccuracy_dt(rsearch,"maximum depth of the tree","Decision Tree + word bigram + whole
dataset")


# In[ ]:

# balance the dataset and repeat

# In[69]:

bigram_word_tdidf["age"]=age
bigram_word_tdidf_10s=bigram_word_tdidf[bigram_word_tdidf["age"]=="10s"][:balance_ratio]
bigram_word_tdidf_20s=bigram_word_tdidf[bigram_word_tdidf["age"]=="20s"][:balance_ratio]
bigram_word_tdidf_30s=bigram_word_tdidf[bigram_word_tdidf["age"]=="30s"][:balance_ratio]
# bigram_word_tdidf_balanced will be the new dataset after balanced
bigram_word_tdidf_balanced=pd.concat([bigram_word_tdidf_10s,bigram_word_tdidf_20s,bigram_word_
tdidf_30s],axis=0)
age_balance=bigram_word_tdidf_balanced["age"].as_matrix()
del bigram_word_tdidf_balanced["age"]


# In[70]:

#split the train test data
word_train, word_test, age_train, age_test =
train_test_split(bigram_word_tdidf_balanced,age_balance, test_size=0.33, random_state=42)
# testing different depth on age
rsearch = RandomizedSearchCV(estimator=dt, param_distributions=param_grid,n_iter=50)
rsearch.fit(word_train,age_train)
# summarize the results of the random parameter search
print("dt word balanced best score on age: ",rsearch.best_score_)
print("dt word balanced best depth on age: ",rsearch.best_estimator_.max_depth)
dt=DecisionTreeClassifier(max_depth=rsearch.best_estimator_.max_depth)
```

```
dt.fit(word_test,age_test)
print("dt word balanced test score on age: ",dt.score(word_test,age_test))

plotAccuracy_dt(rsearch,"maximum depth of the tree","Decision Tree + word bigram + balanced
dataset")
```

Appendix C: Naive Bayes Implementation

```
import pandas as pd
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import math
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.grid_search import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
import numpy as np
from sklearn.naive_bayes import MultinomialNB

# read the labels and bigram words frequency
labels=pd.read_excel("categorizing info.xlsx",header=None)
labels.columns=["filename","gender","age","others"]
age=labels["age"].as_matrix()
gender=labels["gender"].as_matrix()
print(age[:5])
print(gender[:5])
balance_ratio=math.floor(0.07*len(labels))
balance_ratio

# the tdidf words features is for decision tree
bigram_word_tdidf=pd.read_csv("arraywordtfidf.csv",header=None)
len(bigram_word_tdidf)
#len(bigram_char_tdidf)
len(age)
len(gender)
# begin naive bayes
# do the word
bigram_word=pd.read_csv("F://Word_Ngrams/arrayword.csv",header=None)
len(bigram_word)
#len(bigram_char_tdidf)
len(age)
len(gender)

def plotAccuracy_nb(rs,xlabel,title):
    # get the information from the grid research
    depths=list()
    scores = list()
    scores_std = list()
    for e in rs.grid_scores_:
        depths.append(e.parameters.get("alpha"))
        scores.append(e.mean_validation_score)
        scores_std.append(np.std(e.cv_validation_scores))

    # plot out the mean accuracy and standard errors from cross validation of different alphas

    plt.figure().set_size_inches(13, 11)
    plt.semilogx(depths, scores)

    # plot error lines showing +/- std. errors of the scores
    std_error = scores_std / np.sqrt(3)

    plt.semilogx(depths, scores + std_error, 'b--')
    plt.semilogx(depths, scores - std_error, 'b--')
```

```
    # alpha=0.2 controls the translucency of the fill color
    plt.fill_between(depths, scores + std_error, scores - std_error, alpha=0.2)
    plt.ylabel('CV score +/- std error',fontsize=15)
    plt.xlabel(xlabel,fontsize=15)
    plt.title(title,fontsize=20)
    plt.axhline(np.max(scores), linestyle='--', color='.5')
    plt.xlim([depths[0], depths[-1]])
    #plt.ylim([ np.min(scores), np.max(scores)])
    #plt.vlines(rs.best_estimator_.max_depth,ymin=linestyle='dashed')
    fn=title+".png"
    plt.savefig(fn)

#from sklearn.naive_bayes import MultinomialNB
nb=MultinomialNB()
param_grid={"alpha":np.logspace(-2,0,10)}
rsearch = RandomizedSearchCV(estimator=nb,
param_distributions=param_grid,n_iter=len(param_grid))

#split the train test data
word_train, word_test, age_train, age_test = train_test_split(bigram_word, age, test_size=0.33,
random_state=42)
# testing different depth on age

rsearch.fit(word_train,age_train)
# summarize the results of the random parameter search
print("nb word whole best score on age: ",rsearch.best_score_)
print("nb word whole best depth on age: ",rsearch.best_estimator_.alpha)
nb=MultinomialNB(alpha=rsearch.best_estimator_.alpha)
nb.fit(word_test,age_test)
print("nb word whole test score on age: ",nb.score(word_test,age_test))
plotAccuracy_nb(rsearch,"Laplace smoothing parameter","NaiveBayes + word bigram + whole
dataset")

# balance the dataset
bigram_word["age"]=age
bigram_word_10s=bigram_word[bigram_word["age"]=="10s"][:balance_ratio]
bigram_word_20s=bigram_word[bigram_word["age"]=="20s"][:balance_ratio]
bigram_word_30s=bigram_word[bigram_word["age"]=="30s"][:balance_ratio]
# bigram_word_tdidf_balanced will be the new dataset after balanced
bigram_word_balanced=pd.concat([bigram_word_10s,bigram_word_20s,bigram_word_30s],axis=0)
age_balance=bigram_word_balanced["age"].as_matrix()
del bigram_word_balanced["age"]

#split the train test data
word_train, word_test, age_train, age_test = train_test_split(bigram_word_balanced,age_balance,
test_size=0.33, random_state=42)
# testing different depth on age
rsearch.fit(word_train,age_train)
# summarize the results of the random parameter search
print("nb word balanced best score on age: ",rsearch.best_score_)
print("nb word balanced best depth on age: ",rsearch.best_estimator_.alpha)
nb=MultinomialNB(alpha=rsearch.best_estimator_.alpha)
nb.fit(word_test,age_test)
print("nb word balanced test score on age: ",nb.score(word_test,age_test))
plotAccuracy_nb(rsearch,"Laplace smoothing parameter","NaiveBayes + word bigram + balanced
dataset")

# do the char
bigram_char=pd.read_csv("arraychar.csv",header=None,lineterminator=";")
```

```
#split the train test data
char_train, char_test, age_train, age_test = train_test_split(bigram_char, age, test_size=0.33,
random_state=42)
# testing different depth on age

rsearch.fit(char_train,age_train)
# summarize the results of the random parameter search
print("nb char whole best score on age: ",rsearch.best_score_)
print("nb char whole best depth on age: ",rsearch.best_estimator_.alpha)
nb=MultinomialNB(alpha=rsearch.best_estimator_.alpha)
nb.fit(char_test,age_test)
print("nb word whole test score on age: ",nb.score(char_test,age_test))
plotAccuracy_nb(rsearch,"Laplace smoothing parameter","NaiveBayes + char bigram + whole
dataset")


# balance the dataset
bigram_char["age"]=age
bigram_char_10s=bigram_char[bigram_char["age"]=="10s"][:balance_ratio]
bigram_char_20s=bigram_char[bigram_char["age"]=="20s"][:balance_ratio]
bigram_char_30s=bigram_char[bigram_char["age"]=="30s"][:balance_ratio]
# bigram_char_balanced will be the new dataset after balanced
bigram_char_balanced=pd.concat([bigram_char_10s,bigram_char_20s,bigram_char_30s],axis=0)
age_balance=bigram_char_balanced["age"].as_matrix()
del bigram_char_balanced["age"]

#split the train test data
char_train, char_test, age_train, age_test = train_test_split(bigram_char_balanced,
age_balance, test_size=0.33, random_state=42)
# testing different depth on age

rsearch.fit(char_train,age_train)
# summarize the results of the random parameter search
print("nb char _balanced best score on age: ",rsearch.best_score_)
print("nb char _balanced best depth on age: ",rsearch.best_estimator_.alpha)
nb=MultinomialNB(alpha=rsearch.best_estimator_.alpha)
nb.fit(char_test,age_test)
print("nb word _balanced test score on age: ",nb.score(char_test,age_test))
plotAccuracy_nb(rsearch,"Laplace smoothing parameter","NaiveBayes + char bigram + balanced
dataset")
```

Appendix D: Maximum Entropy Implementation

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Load data from CSVs
print("Loading Datafiles")
#features = pd.read_csv('data/arraychartfidf.csv', lineterminator=';')#, nrows=1000)
features = pd.read_csv('data/arraywordtfidf.csv')
age = np.ravel(pd.read_csv('data/age.csv', lineterminator=',', nrows=features.shape[0]))
sex = np.ravel(pd.read_csv('data/sex.csv', lineterminator=',', nrows=features.shape[0]))

# Split data into training and test data
feat_tr, feat_ts, age_tr, age_ts = train_test_split(features, age,
                                test_size=0.33, random_state=42)
feat_tr, feat_ts, sex_tr, sex_ts = train_test_split(features, sex,
                                test_size=0.33, random_state=42)

# Fit models to training data
print("\nTraining Max Entropy Models")
maxent = LogisticRegression()
agemaxent = maxent.fit(feat_tr, age_tr)
maxent = LogisticRegression()
sexmaxent = maxent.fit(feat_tr, sex_tr)

# Test the MaxEnt models
print("Max Entropy Classifier")
print("    Training Accuracy")
print("        Age: ", agemaxent.score(feat_tr, age_tr))
print("        Sex: ", sexmaxent.score(feat_tr, sex_tr))
print("    Test Accuracy")
print("        Age: ", agemaxent.score(feat_ts, age_ts))
print("        Sex: ", sexmaxent.score(feat_ts, sex_ts))
```

Appendix E: k-Nearest Neighbors Implementation

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Load data from CSVs
print("Loading Datafiles")
#features = pd.read_csv('data/arraychartfidf.csv', lineterminator=';')#, nrows=1000)
features = pd.read_csv('data/arraywordtfidf.csv')
age = np.ravel(pd.read_csv('data/age.csv', lineterminator=',', nrows=features.shape[0]))
sex = np.ravel(pd.read_csv('data/sex.csv', lineterminator=',', nrows=features.shape[0]))

# Split data into training and test data
feat_tr, feat_ts, age_tr, age_ts = train_test_split(features, age,
                                test_size=0.33, random_state=42)
feat_tr, feat_ts, sex_tr, sex_ts = train_test_split(features, sex,
                                test_size=0.33, random_state=42)

# Fit KNN model to training data
print("\nTraining KNN Models")
knn = KNeighborsClassifier(n_neighbors=5, leaf_size=30)
ageknn = knn.fit(feat_tr, age_tr)
knn = KNeighborsClassifier(n_neighbors=5, leaf_size=30)
sexknn = knn.fit(feat_tr, sex_tr)

# Test KNN models
print("K-Nearest Neighbors Classifier")
print("    Training Accuracy")
print("        Age: ", ageknn.score(feat_tr, age_tr))
print("        Sex: ", sexknn.score(feat_tr, sex_tr))
print("    Test Accuracy")
print("        Age: ", ageknn.score(feat_ts, age_ts))
print("        Sex: ", sexknn.score(feat_ts, sex_ts))
```

Appendix F: Support Vector Machine Implementation

```
### External Libraries
import csv
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.cross_validation import train_test_split
from sklearn.svm import LinearSVC, SVC
from numpy import linalg

### Custom Kernel Function
def linear(x1, x2):
   return np.dot(x1, x2)

def poly(x, y, p=3):
   return (np.dot(x, y) + 1) ** p

def sigmoid(x, y, s=5.0):
   return np.exp(-linalg.norm(x-y)**2 / (2 * (s ** 2)))

limit = 25439

# Matrix Dimensions
# (25440, 2321) arraywordtfidf.csv
# (25439, 20566) arraychartfidf.csv
# (25440,)      age.csv
# (25440,)      sex.csv

### Reading Files
featureFile = 'arraywordtfidf.csv'
with open(featureFile, 'r') as file:
   features = list(csv.reader(file))
   for row in range(len(features) - 1):
       for col in range(14):
           features[row][col] = float(features[row][col])

features = np.array(features)
features = features[:limit,:]

### Following file import for arraychartfidf
# featureFile = 'arraychartfidf.csv'
# with open(featureFile, 'r') as file:
#     features = pd.read_csv(featureFile, lineterminator=';')
#
# features = np.array(features, dtype=float)
# features = features[:limit,:]


ageFile = 'age.csv'
with open(ageFile, 'r') as file:
   age = list(csv.reader(file))
   for row in range(len(age) - 1):
       for col in range(14):
           age[row][col] = float(age[row][col])

age = np.transpose(age)
age = np.array(age[:limit,0])

sexFile = 'sex.csv'
```

```
with open(sexFile, 'r') as file:
   sex = list(csv.reader(file))
   for row in range(len(sex) - 1):
       for col in range(14):
           sex[row][col] = float(sex[row][col])

sex = np.transpose(sex)
sex = np.array(sex[:limit,0])

### Printing dimensions of the matrices for validation
# print(features.shape, age.shape, sex.shape)


# SVM For linear Kernel
print ("Kernel: Linear")
features_train, features_test, y_train, y_test = train_test_split(features, age,
test_size=0.33, random_state=42)
features_train, features_test, y_train, y_test = np.array(features_train, dtype=float),
np.array(features_test, dtype=float), np.array(y_train, dtype=float), np.array(y_test,
dtype=float)
## Using Sklearn Kernel
svc = LinearSVC()
svc.fit(features_train, y_train)
print ("Training Set, Age: ", svc.score(features_train, y_train))
print ("Testing Set, Age : ", svc.score(features_test, y_test))
## if using linear kernel function
# svc = SVC(kernel='precomputed')
# svc.fit(linear(features_train,features_train.T), y_train)
# print ("Training Set, Age: ", cross_val_score(svc, features_train, y_train).mean())
# print ("Testing Set, Age : ", cross_val_score(svc, features_test, y_test).mean())


features_train, features_test, y_train, y_test = train_test_split(features, sex,
test_size=0.33, random_state=42)
features_train, features_test, y_train, y_test = np.array(features_train, dtype=float),
np.array(features_test, dtype=float), np.array(y_train, dtype=float), np.array(y_test,
dtype=float)
## Using Sklearn Kernel
svc = LinearSVC()
svc.fit(features_train, y_train)
print ("Training Set, Sex: ", svc.score(features_train, y_train))
print ("Testing Set, Sex : ", svc.score(features_test, y_test))
## if using linear kernel function
# svc = SVC(kernel='precomputed')
# svc.fit(linear(features_train,features_train.T), y_train)
# print("Training Set, Sex: ", cross_val_score(svc, features_train, y_train).mean())
# print("Testing Set, Sex : ", cross_val_score(svc, features_test, y_test).mean())
print(" ")

# SVM For RBF Kernel (using sklearn only)
print ("Kernel: RBF")
features_train, features_test, y_train, y_test = train_test_split(features, age,
test_size=0.33, random_state=42)
features_train, features_test, y_train, y_test = np.array(features_train, dtype=float),
np.array(features_test, dtype=float), np.array(y_train, dtype=float), np.array(y_test,
dtype=float)
svc = SVC()
svc.fit(features_train, y_train)
print ("Training Set, Age: ", cross_val_score(svc, features_train, y_train).mean())
print ("Testing Set, Age : ", cross_val_score(svc, features_test, y_test).mean())
```

```
features_train, features_test, y_train, y_test = train_test_split(features, sex,
test_size=0.33, random_state=42)
features_train, features_test, y_train, y_test = np.array(features_train, dtype=float),
np.array(features_test, dtype=float), np.array(y_train, dtype=float), np.array(y_test,
dtype=float)
svc = SVC()
svc.fit(features_train, y_train)
print ("Training Set, Sex: ", cross_val_score(svc, features_train, y_train).mean())
print ("Testing Set, Sex : ", cross_val_score(svc, features_test, y_test).mean())
print (" ")

# SVM for Polynomial kernel
print ("Kernel: Poly")
features_train, features_test, y_train, y_test = train_test_split(features, age,
test_size=0.33, random_state=42)
features_train, features_test, y_train, y_test = np.array(features_train, dtype=float),
np.array(features_test, dtype=float), np.array(y_train, dtype=float), np.array(y_test,
dtype=float)
## Using Sklearn Kernel
svc = SVC(kernel='poly')
svc.fit(features_train, y_train)
## if using poly kernel function
# svc = SVC(kernel='precomputed')
# svc.fit(poly(features_train,features_train.T), y_train)
print ("Training Set, Age: ", cross_val_score(svc, features_train, y_train).mean())
print ("Testing Set, Age : ", cross_val_score(svc, features_test, y_test).mean())


features_train, features_test, y_train, y_test = train_test_split(features, sex,
test_size=0.33, random_state=42)
features_train, features_test, y_train, y_test = np.array(features_train, dtype=float),
np.array(features_test, dtype=float), np.array(y_train, dtype=float), np.array(y_test,
dtype=float)
## Using Sklearn Kernel
svc = SVC(kernel='poly')
svc.fit(features_train, y_train)
## if using poly kernel function
# svc = SVC(kernel='precomputed')
# svc.fit(poly(features_train,features_train.T), y_train)
print ("Training Set, Sex: ", cross_val_score(svc, features_train, y_train).mean())
print ("Testing Set, Sex : ", cross_val_score(svc, features_test, y_test).mean())
print (" ")

# SVM for Sigmoid Kernel
print ("Kernel: Sigmoid")
features_train, features_test, y_train, y_test = train_test_split(features, age,
test_size=0.33, random_state=42)
features_train, features_test, y_train, y_test = np.array(features_train, dtype=float),
np.array(features_test, dtype=float), np.array(y_train, dtype=float), np.array(y_test,
dtype=float)
## Using Sklearn Kernel
svc = SVC(kernel='sigmoid')
svc.fit(features_train, y_train)
## if using poly kernel function
# svc = SVC(kernel='precomputed')
# svc.fit(sigmoid(features_train,features_train.T), y_train)
print ("Training Set, Age: ", cross_val_score(svc, features_train, y_train).mean())
print ("Testing Set, Age : ", cross_val_score(svc, features_test, y_test).mean())
```

```
features_train, features_test, y_train, y_test = train_test_split(features, sex,
test_size=0.33, random_state=42)
features_train, features_test, y_train, y_test = np.array(features_train, dtype=float),
np.array(features_test, dtype=float), np.array(y_train, dtype=float), np.array(y_test,
dtype=float)
## Using Sklearn Kernel
svc = SVC(kernel='sigmoid')
svc.fit(features_train, y_train)
## if using poly kernel function
# svc = SVC(kernel='precomputed')
# svc.fit(sigmoid(features_train,features_train.T), y_train)
print ("Training Set, Sex: ", cross_val_score(svc, features_train, y_train).mean())
print ("Testing Set, Sex : ", cross_val_score(svc, features_test, y_test).mean())
```