PredictingAgeAndGender

April 25, 2019

1 Import libraries, create document class to hold info

```
In [ ]: #  /Data/celebs-usa/female contains 381 texts by females
        #  /Data/celebs-usa/male contains 912 texts by males
        #  /Data/celebs-other-json contains text by

        # Identify birth year as that is a constant, these tweets are from 2011-2018, age rang
        # using birth year, predict age 10-15, 15-20, 20-25, 25-30, 30-35, 35-40, 45-55,55+
        #{'25-34', '35-44', '45-54', '55-64', '65+'}

        from os import listdir, makedirs
        from os.path import isfile, join, splitext, split
        import json
        from collections import Counter
        import ftfy
        import re
        import nltk
        import copy
        from collections import Counter
        import seaborn as sns
        import pandas as pd
        import numpy as np
        import ndjson
        import pickle
        import os
        import jsonlines
        from sklearn.base import BaseEstimator, TransformerMixin


        hashtag_re = re.compile(r"#\w+")
        mention_re = re.compile(r"@\w+")
        url_re = re.compile(r"(?:https?://)?(?:[-\w]+\.)+[a-zA-Z]{2,9}[-\w/#~:;.?+=&%@~]*")

        def preprocess(text):
            p_text = hashtag_re.sub("[hashtag]",text)
            p_text = mention_re.sub("[mention]",p_text)
            p_text = url_re.sub("[url]",p_text)
```

```
    p_text = ftfy.fix_text(p_text)
    return p_text.lower()

tokenise_re = re.compile(r"(\[[^\]]+\]|[-'\w]+|[^\s\w\[']+)") #([]|words|other non-spa
def tokenise(text):
    return tokenise_re.findall(text)


class Document:
    def __init__(self, meta={}):
        self.meta = meta
        self.tokens_fql = Counter() #empty Counter, ready to be added to with Counter.
        self.pos_fql = Counter()
        self.pos_list = [] #empty list for pos tags from running text.
        self.num_tokens = 0

    def extract_features_from_text(self, text):
        p_text = preprocess(text)
        tokens = tokenise(p_text)
        self.num_tokens += len(tokens)
        self.tokens_fql.update(tokens) #updating Counter counts items in list, adding
        pos_tagged = nltk.pos_tag(tokens)
        pos = [tag[1] for tag in pos_tagged]
        self.pos_fql.update(pos)
        self.pos_list.extend(pos)

    def extract_features_from_texts(self, texts): #texts should be iterable text lines
        for text in texts:
            extract_features_from_text(text)

    def average_token_length(self):
        sum_lengths = 0
        for key, value in self.tokens_fql.items():
            sum_lengths += len(key) * value
        return sum_lengths / self.num_tokens

class DocumentProcessor(BaseEstimator, TransformerMixin):
    def __init__(self, process_method):
        self.process_method = process_method

    def fit(self, X, y=None): #no fitting necessary, although could use this to build
        return self

    def transform(self, documents):
        for document in documents:
            yield self.process_method(document)

def get_tokens_fql(document):
```

```python
    return document.tokens_fql

def get_pos_fql(document):
    return document.pos_fql

def get_text_stats(document):
    ttr = len(document.tokens_fql) / document.num_tokens
    return {'avg_token_length': document.average_token_length(), 'ttr': ttr }


def read_list(file):
    with open(file) as f:
        items = []
        lines = f.readlines()
        for line in lines:
            items.append(line.strip())
    return items

fws = read_list("functionwords.txt")

def get_fws_fql(document):
    fws_fql = Counter({t: document.tokens_fql[t] for t in fws})
    #dict comprehension, t: fql[t] is token: freq.
    return +fws_fql

def custom_tokenise(text):
    return tokenise_re.findall(text.lower())

def preprocess(text):
    p_text = hashtag_re.sub("[hashtag]",text)
    p_text = mention_re.sub("[mention]",p_text)
    p_text = url_re.sub("[url]",p_text)
    p_text = ftfy.fix_text(p_text)
    return p_text

def confusion_matrix_heatmap(cm, index):
    cmdf = pd.DataFrame(cm, index = index, columns=index)
    dims = (5, 5)
    fig, ax = plt.subplots(figsize=dims)
    sns.heatmap(cmdf, annot=True, cmap="coolwarm", center=0)
    ax.set_ylabel('Actual')
    ax.set_xlabel('Predicted')
```

2 Reading in Celebrity Data, converting to Document Class and saving to pickle file

```
In [ ]: '''
        This gets the celebrity data and adds the correct gender and ages
        to each json object with text
        '''
        def getCelebData():
            path = '/home/jay/Downloads/pan19-celebrity-profiling-training-dataset-2019-01-31/
            path2 = '/home/jay/Downloads/pan19-celebrity-profiling-training-dataset-2019-01-31
            x=0
            # Read in the twitter text
            data = []
            with jsonlines.open(path) as reader:
                for obj in reader:
                    x+=1
                    print('Reading no ',x)
                    data.append(obj)
                    if len(data) >19999:
                        break;
            # Here the correct labels are identified and paried
            labels = []
            with jsonlines.open(path2) as reader:
                for obj in reader:
                    if obj['gender']!='nonbinary':
                        for d in data:
                            if d['id'] == obj['id']:
                                d['gender'] = obj['gender']
                                d['birthyear'] = obj['birthyear']
            return data

        '''
        This function returns the 5 year group a year of birth resides in, e.g. 1995 is betwee
        '''
        def getYearRange(yearOfBirth):
            YearGroupGap = 5
            for minYear in range(1900,2015,YearGroupGap):
                maxYear = minYear+YearGroupGap
                #print('min: ',minYear,'max: ',maxYear)
                if (yearOfBirth >= minYear) and (yearOfBirth < maxYear):
                    return( str(minYear)+'-'+str(maxYear-1) )
            raise Exception('year of Birth passed in - ' + str(yearOfBirth)+' is not in range


        '''
        This helper function uses the Document class to return a doc
        class for each user with the correct gender, age and tweets
```

```
'''
def getDocument(data):
    try:
        gender    = data['gender']
        birthyear = data['birthyear']
        if data['birthyear'] != 'unknown':
            birthYearRange = getYearRange(data['birthyear'])

        doc = Document({'gender': gender, 'birthyear':birthyear, 'birthyearrange':birth
        for tweet in data['text']:
            doc.extract_features_from_text(tweet)
        return doc
    except:
        print("An exception occurred")

'''
Check if the pickle file exists, if not then create it, else read in
'''
corpus = []
if os.path.exists("/home/jay/Documents/AppliedDataMining/FinalProject/Data/CelebFile")
    with open('/home/jay/Documents/AppliedDataMining/FinalProject/Data/CelebFile', 'rb
        corpus = pickle.load(fp)
    corpus = []
    for i in range(5000):
        path = '/home/jay/Documents/AppliedDataMining/FinalProject/Data/20000Celebs/'
        path += 'Celeb'+str(i)
        with open(path, 'rb') as fp:
            obj = pickle.load(fp)
            if obj is not None:
                corpus.append(obj)
            print('done ',i)


    print('CELEB FILE EXISTS')
else:
    print('CELEB FILE DOES NOT EXISTS, CREATING')
    # Call the function to get the twitter data
    corpus = getCelebData()
    print('Read unprocessed, saving to file')
    #with open('/home/jay/Documents/AppliedDataMining/FinalProject/Data/UnProcessedCel
    #    pickle.dump(corpus, fpc)
    print('Saved to file')
    #For each json object, convert it to a document object
    for i in range(len(corpus)):
        print('Doing Obj Number: ',i)
        corpus[i] = getDocument(corpus[i])

    with open('/home/jay/Documents/AppliedDataMining/FinalProject/Data/CelebFile', 'wb
                pickle.dump(corpus, fp)
            print('CELEB FILE CREATED')
        print('OUT ERE')
```

3 Exploratory Data Analysis

```
In [ ]: corpus = [d for d in corpus if d.meta['gender'] != 'nonbinary']
        #Get all the birth years and plot a histogram
        birth_year_y = [d.meta['birthyear'] for d in corpus]
        x = pd.Series(birth_year_y, name="Birth Year")
        sns.distplot(x)

        #Get a count of of the birth years and plot a bar chart
        df = pd.DataFrame.from_dict(Counter(birth_year_y), orient='index').reset_index()
        df.columns = ['Year','Frequency of people born']
        df = df.sort_values(by=['Year'])
        df.plot.bar(x='Year', y='Frequency of people born', rot=90,figsize=(10,10), title='The
        #BirthYearDF = copy.deepcopy(df)


        #Get all the birth years and plot a histogram
        birth_year_y = [d.meta['birthyearrange'] for d in corpus]
        df1 = pd.DataFrame.from_dict(Counter(birth_year_y), orient='index').reset_index()
        df1.columns = ['Year','Frequency of people born']
        df1 = df1.sort_values(by=['Year'])
        df1.plot.bar(x='Year', y='Frequency of people born', rot=90,figsize=(10,10), title='Th


        #Get all the genders and plot a bar chart
        gender_y = [d.meta['gender'] for d in corpus]
        df2 = pd.DataFrame.from_dict(Counter(gender_y), orient='index').reset_index()
        df2.columns = ['Gender','Frequency of Gender']
        df2.plot.bar(x='Gender', y='Frequency of Gender', rot=90,figsize=(10,10), title='The nu
```

4 Get Train and Test Split + Resample

```
In [ ]: from sklearn.model_selection import train_test_split
        from imblearn.under_sampling import RandomUnderSampler

        '''
        Undersample men
        '''
        femaleCorpus = [d for d in corpus if d.meta['gender'] == 'female']
        maleCorpus   = [d for d in corpus if d.meta['gender'] == 'male']
        genderCorpus = maleCorpus[:len(femaleCorpus)] + femaleCorpus

        #Getting gender Train and Test
```

```python
gender_y = [d.meta['gender'] for d in genderCorpus]
gender_X = genderCorpus
Gender_X_train, Gender_X_test, Gender_y_train, Gender_y_test = train_test_split(gender_
genderCorpus = [d for d in corpus if d.meta['gender'] != 'nonbinary']

'''
Calculate the average year and undersample ages so all the ages that are overrepresent
then they are decreased to the average frequency
'''
BirthYearDF = df.reset_index(drop=True)
averageFrequency = round(BirthYearDF['Frequency of people born'].mean())
BirthYearsThatNeedUnderSampling = BirthYearDF.loc[BirthYearDF['Frequency of people born
BirthYearsThatNeedUnderSampling = BirthYearsThatNeedUnderSampling.set_index('Year') #.
BirthYearsThatNeedUnderSampling = BirthYearsThatNeedUnderSampling.to_dict()
BirthYearsThatNeedUnderSampling = BirthYearsThatNeedUnderSampling.get('Frequency of peo

itemsToDelete = []
for c in corpus:
    itemBirthYear = c.meta['birthyear']
    if itemBirthYear in BirthYearsThatNeedUnderSampling:
        FrequencyOfRow = BirthYearsThatNeedUnderSampling[itemBirthYear]
        if FrequencyOfRow > averageFrequency:
            itemsToDelete.append(c)
            BirthYearsThatNeedUnderSampling[itemBirthYear] -= 1
            print('Removed one ',itemBirthYear)
print('Done')


corpus = [celeb for celeb in corpus if celeb not in itemsToDelete]

#Getting Birth_year_range Train and Test
birth_year_y = [d.meta['birthyearrange'] for d in corpus]
birth_year_y = [d.meta['birthyear'] for d in corpus]
birth_year_X = corpus
Birth_X_train, Birth_X_test, Birth_y_train, Birth_y_test = train_test_split(birth_year_
genderCorpus = [d for d in corpus if d.meta['gender'] != 'nonbinary']




GenderCount = Counter(gender_y)
BirthYearCount = Counter(birth_year_y)
print(GenderCount)
print('---------------')
print(BirthYearCount)
```

5 Model Selection, GridSearch to identify best classifier and best params

```
In [ ]: from sklearn.neural_network import MLPClassifier, MLPRegressor
        from sklearn.linear_model import LogisticRegression
        from sklearn.feature_selection import SelectKBest, chi2
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.pipeline import Pipeline ,FeatureUnion
        from sklearn.feature_extraction import DictVectorizer
        from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
        import matplotlib.pyplot as plt
        from sklearn.model_selection import GridSearchCV
        from sklearn.model_selection import cross_validate, StratifiedKFold
        #from sklearn.naive_bayes import MultinomialNB
        from sklearn.svm import SVR
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.ensemble import AdaBoostRegressor, AdaBoostClassifier, VotingClassifier,
        from sklearn.metrics import mean_squared_error
        from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_squared_log_e
        #from sklean.metrics import metrics
        import math
        from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, GradientBoo
        from sklearn import linear_model
        from sklearn.svm import SVC
        #from sklearn.feature_extraction import FeatureHasher
        import pickle


        model = Pipeline([
            ('union', FeatureUnion(
                transformer_list = [
                    ('word', Pipeline([
                        ('processor', DocumentProcessor(process_method = get_pos_fql)),
                        ('vectorizer', DictVectorizer()),
                    ])),
                ],
            )),
            ('clf', None), # to be set by grid search.
        ])

        param_grid={ 'clf': [LogisticRegression(solver='liblinear', random_state=0)
                             ,MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True),
                             RandomForestClassifier(),
                            MLPClassifier(max_iter=400)
                            ],

                    'union__word__processor__process_method': [get_tokens_fql, get_fws_fql, get
```

```
search = GridSearchCV(model, cv = StratifiedKFold(n_splits=5, random_state=0),
                      return_train_score = False,
                      scoring = ['accuracy', 'precision_weighted', 'recall_weighted',
                      refit = 'f1_weighted',
                      param_grid = param_grid
                     )



print('Fitting Clf')
search.fit(Gender_X_train, Gender_y_train)
print('Getting Predictions')
predictions = search.predict(Gender_X_test)

print("Accuracy: ", accuracy_score(Gender_y_test, predictions))
print(classification_report(Gender_y_test, predictions))
print(confusion_matrix(Gender_y_test, predictions))

confusion_matrix_heatmap(confusion_matrix(Gender_y_test,predictions), ['M','F'])
```

6 Predicting Birth Year

```
In [ ]: clf = DecisionTreeRegressor()
        regr = linear_model.LinearRegression()
        reg = linear_model.BayesianRidge()

        no_estimators= 10

        #Ada boost SVR with pos_fql
        clf = SVR(gamma='scale', C=1.0, epsilon=0.2)
        AdaBoostSVR = BaggingRegressor(clf, n_estimators=no_estimators, random_state=0)
        Pipeline_AdaBoostSVR_get_pos_fql = Pipeline([
            ('processor', DocumentProcessor(process_method = get_fws_fql)),
            ('vectorizer', DictVectorizer()),
            ('clf', AdaBoostSVR),
        ])

        #Ada boost Random Forest with pos_fql
        clf = RandomForestRegressor(n_estimators=100)
        AdaBoostRandomForest = BaggingRegressor(clf, n_estimators=no_estimators, random_state=
        Pipeline_AdaBoostRandomForest_get_pos_fql = Pipeline([
            ('processor', DocumentProcessor(process_method = get_fws_fql)),
            ('vectorizer', DictVectorizer()),
            ('clf', AdaBoostRandomForest),
        ])
```

```
#Ada boost Bayesian Ridge with pos_fql
clf = linear_model.BayesianRidge()
AdaBoostBayesianRidge = BaggingRegressor(clf, n_estimators=no_estimators, random_state=
Pipeline_AdaBoostBayesianRidge_get_pos_fql = Pipeline([
    ('processor', DocumentProcessor(process_method = get_pos_fql)),
    ('vectorizer', DictVectorizer()),
    ('clf', AdaBoostRandomForest),
])

#Ada boost SVR with pos_fql
clf = SVR(gamma='scale', C=1.0, epsilon=0.2)
AdaBoostSVR = BaggingRegressor(clf, n_estimators=no_estimators, random_state=0)
Pipeline_AdaBoostSVR_get_fws_fql = Pipeline([
    ('processor', DocumentProcessor(process_method = get_pos_fql)),
    ('vectorizer', DictVectorizer()),
    ('clf', AdaBoostSVR),
])


'''
Creating Master Pipeline to fit each ensemble and average the result
'''
PipeLineList = [Pipeline_AdaBoostRandomForest_get_pos_fql,
                Pipeline_AdaBoostSVR_get_pos_fql,
                Pipeline_AdaBoostSVR_get_fws_fql,
                Pipeline_AdaBoostBayesianRidge_get_pos_fql]

predictions= [0] * len(Birth_X_test)
x = 0
print('Fitting clfs')
for pipeline in PipeLineList:
    x+=1
    print('Fitting Clf number ',x)
    pipeline.fit(Birth_X_train, Birth_y_train)
    print('Getting Predictions ', x)
    predictions += pipeline.predict(Birth_X_test)

predictions = predictions / len(PipeLineList)
```

7 Evaluating Birth Year Classifier

```
In [ ]: print('RMSE: ', math.sqrt(mean_squared_error(predictions,Birth_y_test)))
        print('MSLE: ', mean_squared_log_error(predictions,Birth_y_test))
        print('R2 Score: ', r2_score(predictions,Birth_y_test))
        print('MAE: ', mean_absolute_error(predictions,Birth_y_test))


        res = pd.DataFrame( data = {'Predictions': predictions, 'Actual': Birth_y_test} )
        res[:100].plot( colormap='Paired')

        print("Accuracy: ", accuracy_score(Birth_y_test, predictions))
        print(classification_report(Birth_y_test, predictions))
        print(confusion_matrix(Birth_y_test, predictions))


        #labels = list(set(Birth_y_test+predictions))

        confusion_matrix_heatmap(confusion_matrix(Birth_y_test,predictions)  )
```

8 Predicting Gender

```
In [ ]: no_estimators= 1

        #Ada boost SVM with pos_fql
        SVM = SVC(probability=True, kernel='linear', verbose=True)
        Pipeline_SVM_get_pos_fql = Pipeline([
            ('processor', DocumentProcessor(process_method = get_tokens_fql)),
            ('vectorizer', DictVectorizer()),
            ('clf', SVM),
        ])

        #Ada boost Random Forest with pos_fql
        MLP = MLPClassifier(random_state=0, verbose=1, max_iter=50)
        Pipeline_MLP_get_pos_fql = Pipeline([
            ('processor', DocumentProcessor(process_method = get_tokens_fql)),
            ('vectorizer', DictVectorizer()),
            ('clf', MLP),
        ])

        #Ada boost Random Forest with pos_fql
        Pipeline_MLP_get_FWS_fql = Pipeline([
            ('processor', DocumentProcessor(process_method = get_fws_fql)),
            ('vectorizer', DictVectorizer()),
            ('clf', MLP),
        ])


        #Ada boost Random Forest with pos_fql
        LR = LogisticRegression(verbose=1)
        Pipeline_LR_get_FWS_fql = Pipeline([
            ('processor', DocumentProcessor(process_method = get_fws_fql)),
            ('vectorizer', DictVectorizer()),
            ('clf', LR),
        ])


        #Ada boost Bayesian Ridge with pos_fql
        AdaBoostLogisticRegression = LogisticRegression(verbose=1)
        Pipeline_LR_get_pos_fql = Pipeline([
            ('processor', DocumentProcessor(process_method = get_tokens_fql)),
            ('vectorizer', DictVectorizer()),
            ('clf', AdaBoostLogisticRegression),
        ])

        #Ada boost SVR with fws_fql
        RandForest = RandomForestClassifier(n_estimators=100, verbose=1)
        Pipeline_RandForest_get_fws_fql = Pipeline([
            ('processor', DocumentProcessor(process_method = get_tokens_fql)),
            ('vectorizer', DictVectorizer()),
            ('clf', RandForest),
        ])

        GenderPipeline = VotingClassifier(
            estimators=[('MLP', Pipeline_MLP_get_pos_fql),
                        ('MLP2', Pipeline_MLP_get_FWS_fql),
                        ('LR', Pipeline_LR_get_pos_fql),
                        ('LR1', Pipeline_LR_get_pos_fql),
                        ('LR2', Pipeline_LR_get_pos_fql),
                        ('LR3', Pipeline_LR_get_FWS_fql),
                        ('randfor', Pipeline_RandForest_get_fws_fql)
                       ],
                        voting='hard')
```

9 Evaluating Gender Classifier

```
In [ ]: print("Accuracy: ", accuracy_score(Gender_y_test, predictions))
        print(classification_report(Gender_y_test, predictions))
        print(confusion_matrix(Gender_y_test, predictions))

        confusion_matrix_heatmap(confusion_matrix(Gender_y_test, predictions), ['Male', 'Female

        res = pd.DataFrame( data = {'Predictions': predictions, 'Actual': Gender_y_test} )

        res = res.replace(['male', 'female'], [1, 0])

        res[:20].plot( colormap='Paired')

        math.sqrt(mean_squared_error(predictions,Gender_y_test))
```

10 Creating Stacked Generalisation Meta Classifier Training Data

```
In [ ]: '''
        # The stacked generalisation meta classifier (SGMC) is a combination of the gender ens

        #First the outputs of the of the predicted birth year from each of the bagging regress

        #Then the output of predicted gender needs to be saved, along with the true birth year

        So now a dataset with the following structure is created, where X are the features
        (outputs of the ensembles) and Y is the true birth year.

           X                          Y
        1995,1994,1997,1999,1       1995
        ..................................
        .................................
        .................................
        .................................
        ............


        Once this has been done, the MLP Regressor can then be trained
        using the stacked generalisation method of hold one out

        StackedGeneralisationData = pd.DataFrame({"BirthYearPrediction1":[],
                                                  "BirthYearPrediction2":[],
                                                  "BirthYearPrediction3":[],
                                                  "BirthYearPrediction4":[],
                                                  "GenderPrediction":[]
                                                  "TrueValue":[]
                                                 })
        '''

        #Retrieving saved models from file
        GenderEnsemblePath = '/home/jay/Documents/AppliedDataMining/FinalProject/Classifiers/Ge
        BirthYearEnsemblePath = '/home/jay/Documents/AppliedDataMining/FinalProject/Classifiers
        PipeLineList      = pickle.load(open(BirthYearEnsemblePath, 'rb'))
        GenderPipeline    = pickle.load(open(GenderEnsemblePath, 'rb'))

        #File is created to save SGMC training data
        StackedGeneralisationTrainingData = '/home/jay/Documents/AppliedDataMining/FinalProjec

        x = 0
        for observation in corpus:
                x+=1
                #First the outputs of the of the predicted birth year from each of the bagging
                print('Getting Predictions ', x)
                BirthYearPrediction1 = PipeLineList[0].predict([observation])[0]
                BirthYearPrediction2 = PipeLineList[1].predict([observation])[0]
                BirthYearPrediction3 = PipeLineList[2].predict([observation])[0]
                BirthYearPrediction4 = PipeLineList[3].predict([observation])[0]
                GenderPrediction = GenderPipeline.predict([observation])[0]
                if GenderPrediction == 'male':
                    GenderPrediction = 1
                else:
                    GenderPrediction = 0

                true_value = observation.meta['birthyear']

                # Creating the first Dataframe using dictionary
                StackedGeneralisation_Append = pd.DataFrame({"BirthYearPrediction1":[BirthYearI
                                                        "BirthYearPrediction2":[BirthYearPredic
                                                        "BirthYearPrediction3":[BirthYearPredic
                                                        "BirthYearPrediction4":[BirthYearPredic
                                                        "GenderPrediction":[GenderPrediction],
                                                        "TrueValue":[true_value]
                                                       })
                #Saving data to file
                with open(StackedGeneralisationTrainingData, 'a') as f:
                    StackedGeneralisation_Append.to_csv(f, header=False)
                print('done ', x )
```

11 Training SGMC and Getting Predictions

```
In [ ]: from sklearn.model_selection import train_test_split
        import pandas as pd
        from sklearn.neural_network import MLPRegressor
        StackedGeneralisationTrainingDataPath = '/home/jay/Documents/AppliedDataMining/FinalPro

        StackedGeneralisationTrainingData = pd.read_csv(StackedGeneralisationTrainingDataPath)
        StackedGeneralisationTrainingData = StackedGeneralisationTrainingData.reset_index(drop=

        X = StackedGeneralisationTrainingData[['BirthYearPrediction1','BirthYearPrediction2','I
        y = StackedGeneralisationTrainingData[['TrueValue']]

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state =

        X_train = X_train.reset_index(drop=True)
        X_test = X_test.reset_index(drop=True)
        y_train = y_train.reset_index(drop=True)
        y_test = y_test.reset_index(drop=True)

        #SGMC = RandomForestRegressor(n_estimators=300)
        #SGMC = MLPRegressor()
        SGMC = linear_model.LinearRegression()
        print('Training')
        SGMC.fit(X_train,y_train)
        print('Predicting')
        predictions = SGMC.predict(X_test)
        print('Done')
```

12 Evaluating Stacked Generalisation Meta Classifier

```
In [ ]: print('RMSE: ', math.sqrt(mean_squared_error(predictions,y_test)))
        print('MSLE: ', mean_squared_log_error(predictions,y_test))
        print('R2 Score: ', r2_score(predictions,y_test))
        print('MAE: ', mean_absolute_error(predictions,y_test))
        res = pd.DataFrame( data = {'Predictions': predictions, 'Actual': y_test['TrueValue']}
        res[:100].plot( colormap='Paired')
```

13 Testing The SGMC and Gender Ensemble on Tweets from Family + Friends

```
In [ ]: #Testing on friends and family
        import pickle
        GenderEnsemblePath = '/home/jay/Documents/AppliedDataMining/FinalProject/Classifiers/Ge
        BirthYearEnsemblePath = '/home/jay/Documents/AppliedDataMining/FinalProject/Classifiers
        PipeLineList      = pickle.load(open(BirthYearEnsemblePath, 'rb'))
        GenderPipeline    = pickle.load(open(GenderEnsemblePath, 'rb'))

        path = 'alishapatel28_tweets.json'
        #path = 'ashnapatel_tweets.json'
        #path = 'paytonmmusic_tweets.json'

        def SGMC_Predict(observation):
            BirthYearPrediction1 = PipeLineList[0].predict([observation])[0]
            BirthYearPrediction2 = PipeLineList[1].predict([observation])[0]
            BirthYearPrediction3 = PipeLineList[2].predict([observation])[0]
            BirthYearPrediction4 = PipeLineList[3].predict([observation])[0]
            GenderPrediction = GenderPipeline.predict([observation])[0]
            if GenderPrediction == 'male':
                GenderPrediction = 1
            else:
                GenderPrediction = 0

            # Creating the first Dataframe using dictionary
            StackedGeneralisation_Append = pd.DataFrame({"BirthYearPrediction1":[BirthYearPredi
                                                        "BirthYearPrediction2":[BirthYearPredi
                                                        "BirthYearPrediction3":[BirthYearPredi
```

```
                                                "BirthYearPrediction4":[BirthYearPredi
                                                "GenderPrediction":[GenderPrediction],

                                               })
    return SGMC.predict(StackedGeneralisation_Append)


def getTestData():
    # Read in the twitter text
    data = []
    with open(path) as json_file:
        data = json.load(json_file)

    return data

def getTestDocument(data):
    try:
        doc = Document({}) #include metadata
        for tweet in data:
            doc.extract_features_from_text(tweet['full_text'])
        return doc
    except:
        print("An exception occurred")

TestCorpus = getTestData()
TestCorpus = getTestDocument(TestCorpus)
PredictedBirthYear = 0
PredictedGender = ''
PredictedBirthYear = SGMC_Predict(TestCorpus)
PredictedGender = GenderPipeline.predict([TestCorpus])
print('Predicted Birth-Year for: ', path, 'is: ', PredictedBirthYear)
print('Predicted Age for ', path, 'is: ', 2018 - PredictedBirthYear )
print('Predicted Gender for: ', path, 'is: ', PredictedGender)
```

In [ ]: