Classifying Subreddit Post Origins with NLP Pipelines.

Timothy L. Carter

Getting Started

- Pushift API
  - Python
- Scikit Learn
  - NLTK

Special Thanks

~

Jason Michael Baumgartner

Streamlining Cleaning

Practicing Pythonic

```
def _clean_tokens(corpus,column):

    my_stop_words = ['https','com','www','people','know','actually',
                     'world','time','years','fact','facts','fake','like',
                     'sk','10','en','day','water','did','just','the']

    stop_words = text.ENGLISH_STOP_WORDS.union(my_stop_words)

    for row in corpus[column]:
        for word in row:
            if word in stop_words:
                row.remove(word)

    return corpus
```

```
def _string_tokens(corpus,column):
    for i , row in enumerate(corpus[column]):
        corpus[column][i]=' '.join(corpus[column][i])
    return corpus
```

```
def make_clean_string_corpus(corpus,column):

    corpus = _make_tokens(corpus,column)
    corpus = _clean_tokens(corpus, column)
    corpus = _string_tokens(corpus,column)

    return corpus
```

```
# diplays preview of dataframe for checking changes
def disp_hud(hud):
    base_group = corpus.groupby(['fact']).mean()
    head = corpus.head(2)

    hud = [base_group,head]
    disp = ['mean','preview']

    for i,li in enumerate(hud):
        print(disp[i])
        display(li)
#disp_hud(hud)
```

```
def _make_tokens(corpus,column):
    tokenizer = RegexpTokenizer('\w+/\$[\d\.]+/\S+')
    corpus[column] = [tokenizer.tokenize(row) for row in corpus[column]]
    return corpus
```

Practicing Pythonic

Pipeline Processing

```
pipelines = [
    ("logreg_Lasso_1_cv", logreg_Lasso_1_cv),
    ("logreg_Lasso_10_cv", logreg_Lasso_10_cv),
    ("logreg_Ridge_1_cv", logreg_Ridge_1_cv),
    ("logreg_Ridge_10_cv", logreg_Ridge_10_cv),
    ("mnb_cv", mnb_cv),
    ("mnb_tf", mnb_tf),
    ("nb_cv", nb_cv),
    ("nb_tf", nb_cv),
    ("svm_cv", svm_cv),
    ("svm_tf", svm_tf)
]
```

```
def fit_pipes():
    for name, model in pipelines:
        model.fit(X_train,y_train)
```

```
def score_pipes(X,y):
    scores = []
    for name, model in pipelines:
        scores.append(model.score(X,y))
        print( f'{name} Accuracy: {model.score(X,y)} ' )
    df = pd.DataFrame(columns = ['scores, models'], index = None)

    return scores
```

```
def get_gap(train_score,test_score):
    output = []
    for i , li in enumerate(train_scores):
        dif =  li - test_scores[i]
        output.append(dif)
    result = pd.DataFrame()
    result['Model'] = [name for name , model in pipelines]
    result['Gap'] = [li for li in output]

    return result
```

```
def best_score(X,y):
    best_accuracy = 0.0
    best_classifier = ''
    best_pipeline = ''



    for name, model in pipelines:
        if model.score(X,y) > best_accuracy:
            best_accuracy = model.score(X_test, y_test)
            best_pipeline = model
            best_classifier = name

    print(f' Best Accuracy: {best_accuracy}')
    print(f'Model: {best_classifier}')
```

Table of Contents

01

A Sink Full of Sinks

50/50

The Score To Beat

Lion’s Mane Jellyfish have triple helix shaped DNA.

Fortnite was created by the Catholic church to stop people having sex before marriage

Libraries in this world contain a lot of Books bound in Human flesh.

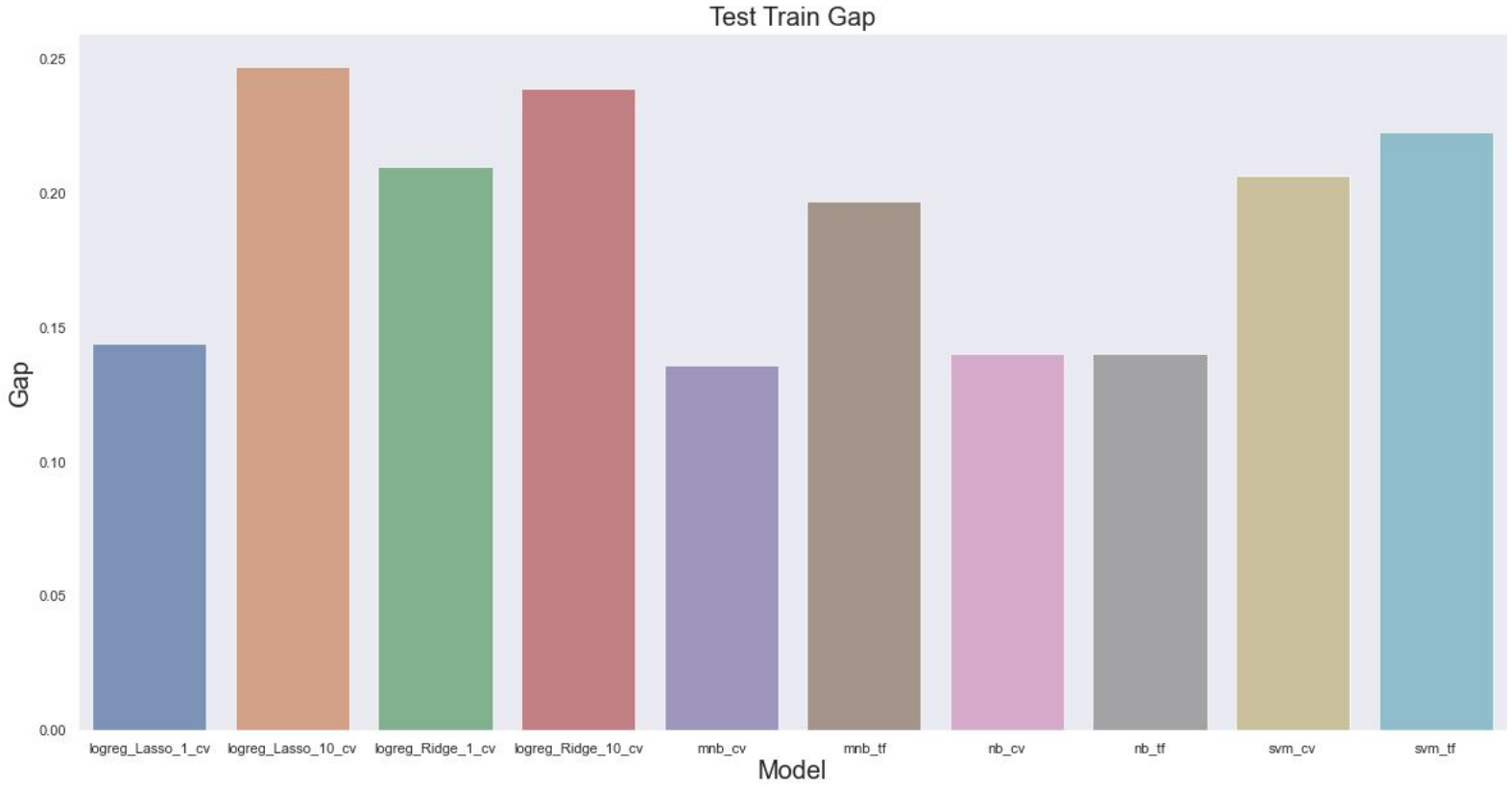
Only female mosquitoes bite. They need the blood to reproduce. Male mosquitoes eat flower nectar.

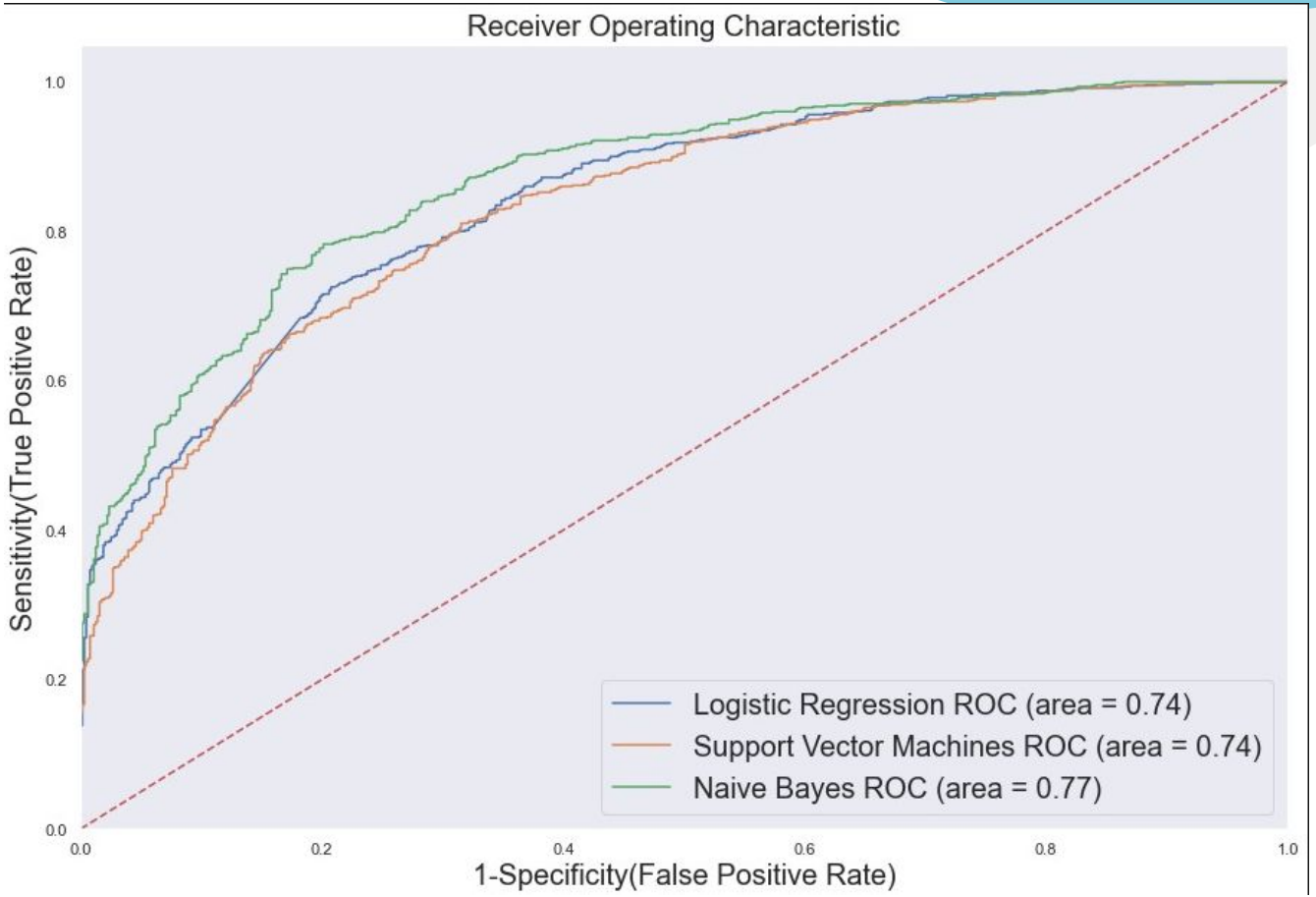
Starting Out

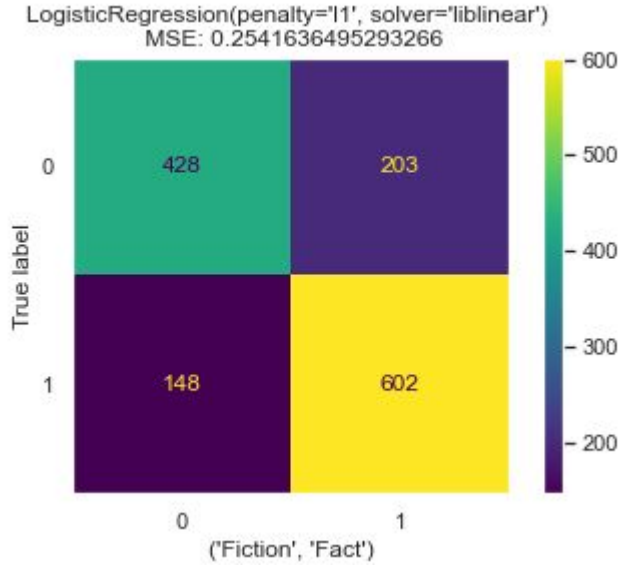
85% Train - Logistic Regression

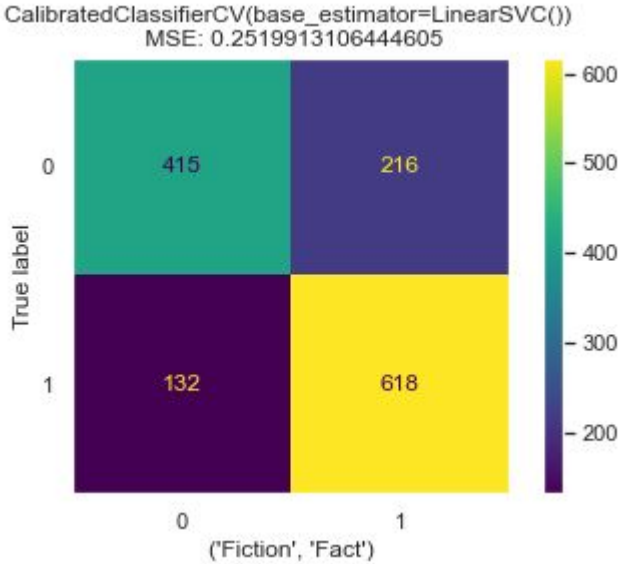
68% Test - SVM wins again!

17% Min. Dif. - Naive Bayes

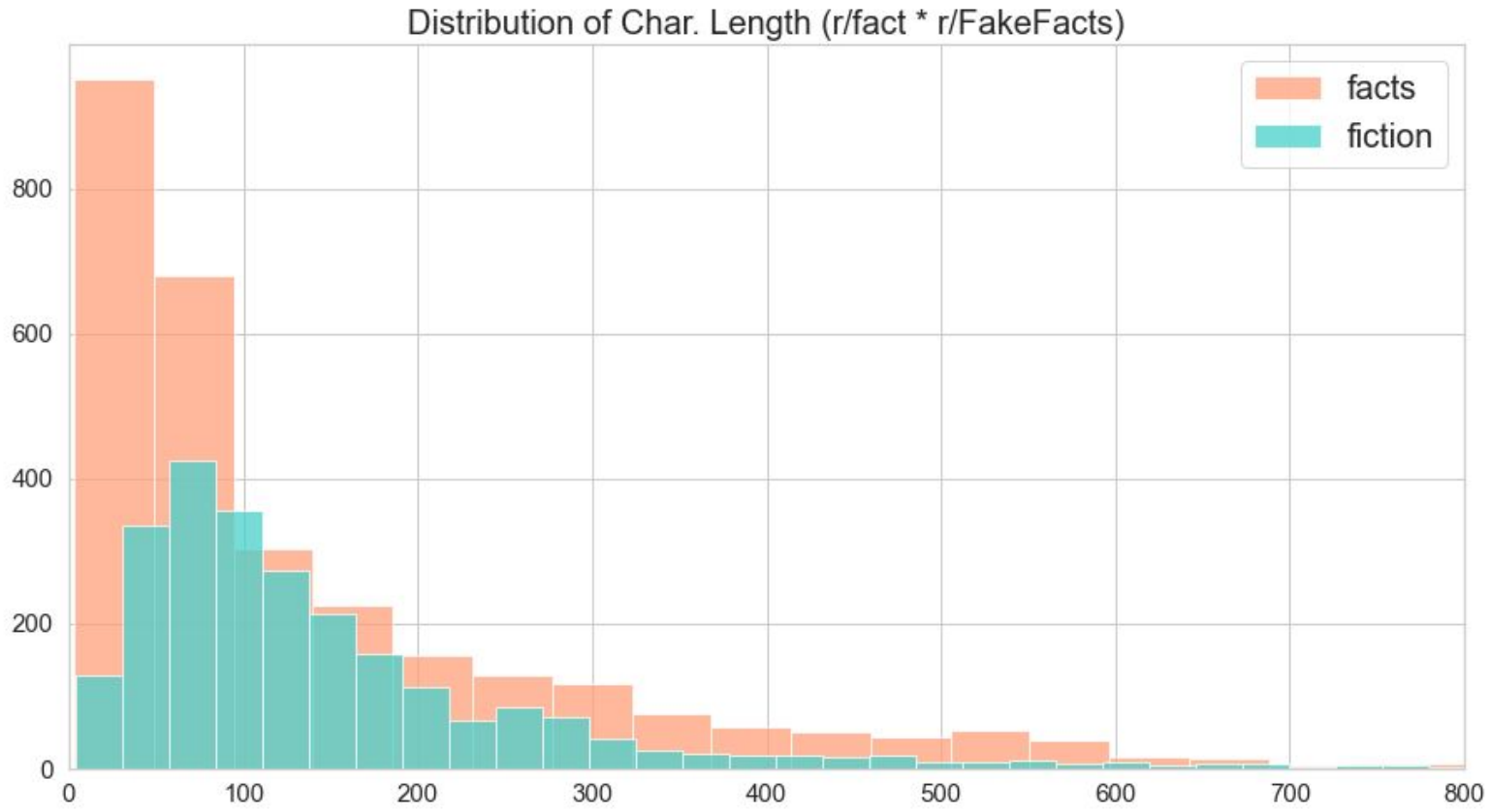

Test Train Gap
Gap
0.25
0.20
0.15
0.10
0.05
0.00
logreg_Lasso_1_cv
logreg_Lasso_10_cv
logreg_Ridge_1_cv
logreg_Ridge_10_cv
mnb_cv
mnb_tf
nb_cv
nb_tf
svm_cv
svm_tf
Model


Receiver Operating Characteristic
Sensitivity(True Positive Rate)
1-Specificity(False Positive Rate)
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression ROC (area = 0.74)
Support Vector Machines ROC (area = 0.74)
Naive Bayes ROC (area = 0.77)

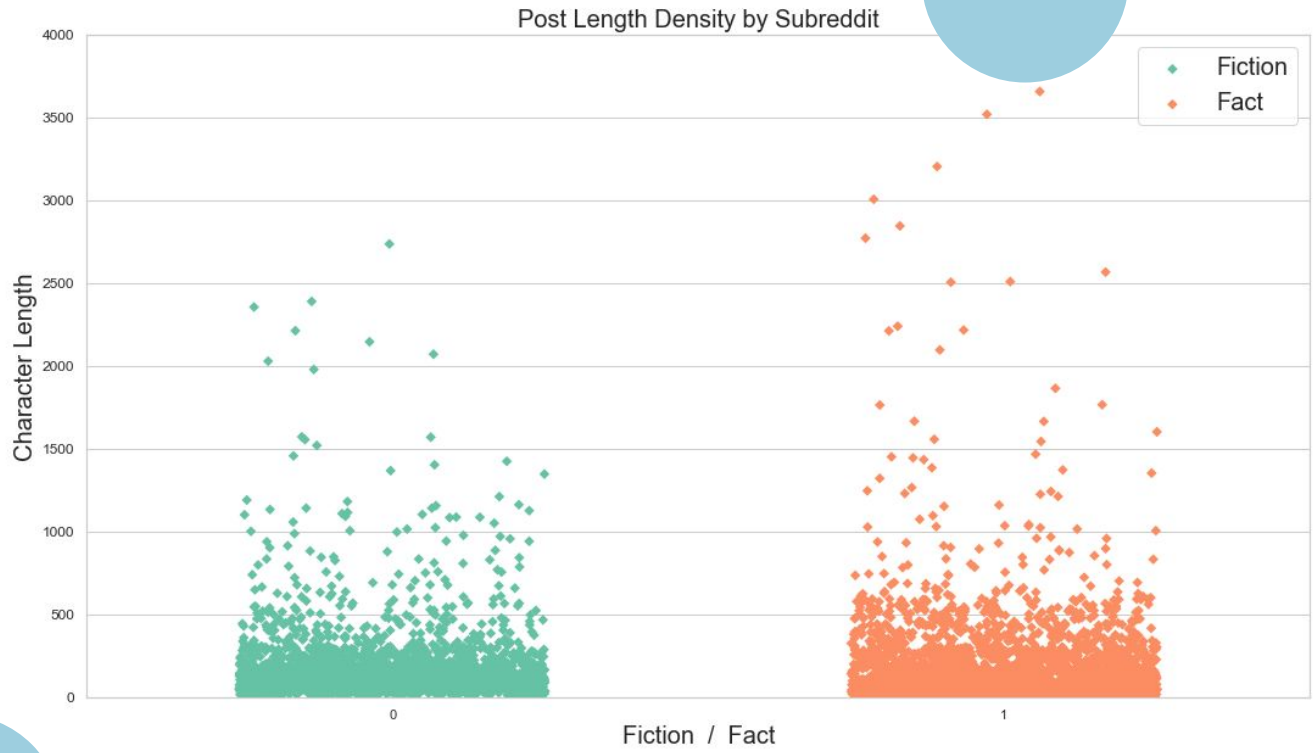

LogisticRegression(penalty='l1', solver='liblinear')
MSE: 0.2541636495293266
True label
0
1
428
203
148
602
0
1
('Fiction', 'Fact')
600
500
400
300
200


CalibratedClassifierCV(base_estimator=LinearSVC())
MSE: 0.2519913106444605
True label
0
1
415
216
132
618
0
1
('Fiction', 'Fact')
600
500
400
300
200

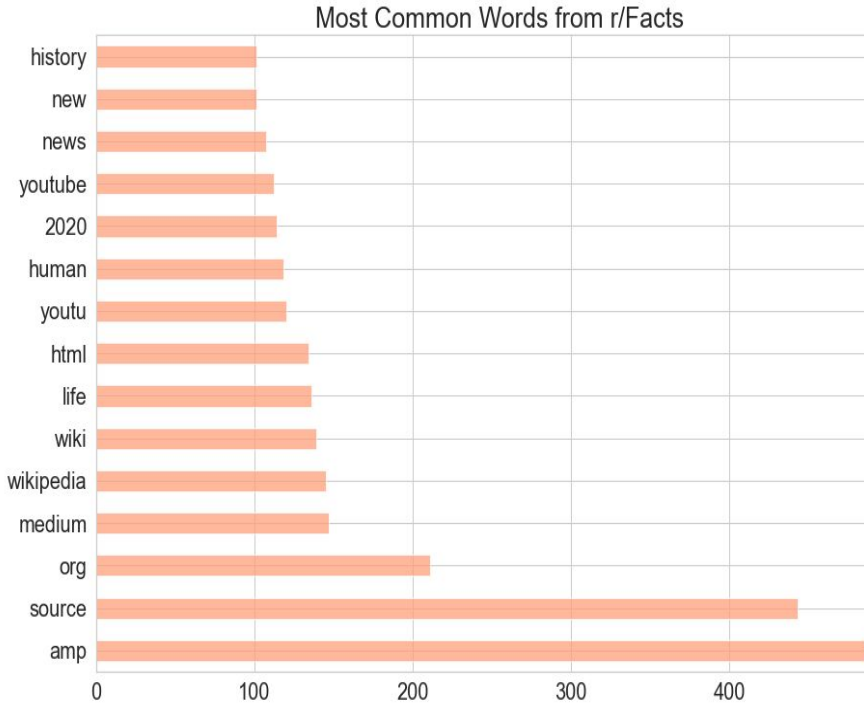
02
Exploring The Lexicon

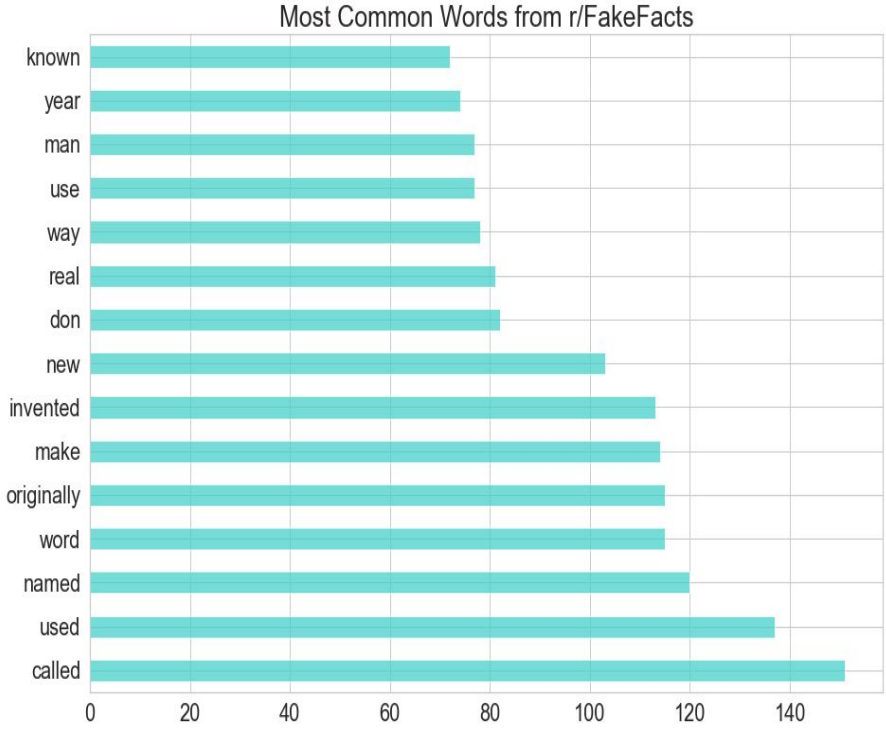

Distribution of Char. Length (r/fact * r/FakeFacts)
facts
fiction
800
600
400
200
0
0
100
200
300
400
500
600
700
800

Let's Take a Look



Large Lot & Central Air





Most Common Words

r/Facts

1. Amp
2. Source
3. Org
4. Wikipedia
5. Wiki
6. Life
7. Html
8. Human
9. Youtube
10. History
11. Medium
12. news

Corpus

1. Amp
2. Source
3. Called
4. Used
5. Org
6. Make
7. New
8. Life
9. Word
10. Human
11. Year
12. Named
13. Man

r/FakeFacts

1. Called
2. Used
3. Originally
4. Invented
5. New
6. Real
7. Year
8. Known
9. Man
10. Word
11. Way
12. Named

03

Pipeline's

Mean Word Embeddings (W2V)

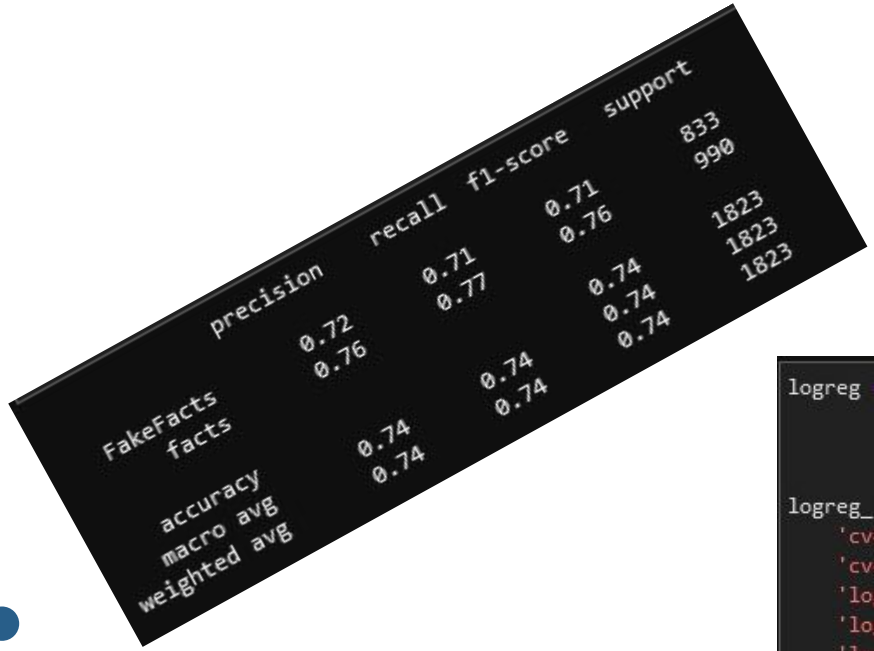
```
with open("glove.6B.50d.txt", "rb") as lines:
    w2v = {line.split()[0]: np.array(map(float, line.split()[1:]))
           for line in lines}
```

```
class MeanEmbeddingVectorizer(object):
    def __init__(self, word2vec):
        self.word2vec = word2vec
        if len(word2vec)>0:
            self.dim=len(word2vec[next(iter(glove_small))])
        else:
            self.dim=0

    def fit(self, X, y):
        return self

    def transform(self, X):
        return np.array([
            np.mean([self.word2vec[w] for w in words if w in self.word2vec]
                    or [np.zeros(self.dim)], axis=0)
            for words in X
        ])
```

Logistic Regression



	precision	recall	f1-score	support
FakeFacts	0.72	0.71	0.71	833
facts	0.76	0.77	0.76	990
accuracy			0.74	1823
macro avg	0.74	0.74	0.74	1823
weighted avg	0.74	0.74	0.74	1823

```
logreg = Pipeline([
        ('cvec', CountVectorizer()),
        ('logreg',LogisticRegression())]);

logreg_params = {
    'cvec__max_df': np.linspace(0.001,1,10),
    'cvec__min_df': np.linspace(0.001,1,10),
    'logreg__penalty': ['l1','l2'],
    'logreg__dual': [True,False],
    'logreg__C': np.linspace(0.001,1,10),
    'logreg__solver': ['newton-cg','lbfgs','liblinear','sag','saga'],
    'logreg__max_iter': [1_000,2_000,3_000]
}


grid = GridSearchCV(logreg, logreg_params, cv=5, n_jobs = 6, verbose = 2)

%time
grid.fit(X_train,y_train)
```

Tuning Model

76% Train - Support Vectors

76% Test _ SVM wins again!

14% Min. Dif.
Multinomial Naive Bayes

Next Steps

The results of this model will be used as a feature in the model of our final product.

Develop Sentiment Analysis .

Grow Dataset , Iterate Cleaning .