```
# setup
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
import seaborn as sb
from sklearn.naive_bayes import MultinomialNB
import math

from sklearn.linear_model import LogisticRegression

# read data
df = pd.read_csv('name_gender_dataset.csv')
df = df.fillna(0)

df.head()
```

	Name	Gender	Count	Probability
0	James	M	5304407	0.014517
1	John	M	5260831	0.014398
2	Robert	M	4970386	0.013603
3	Michael	M	4579950	0.012534
4	William	M	4226608	0.011567

```
# get aggregate count
namegraph = df.groupby(['Name', 'Gender'], as_index=False)['Count'].sum()
namegraph.head(20)
```

	Name	Gender	Count
0	A	F	2
1	A	M	2
2	A'Aff	F	1
3	A'Aron	M	1
4	A'Dele	F	1
5	A'Isha	F	1
6	A'Ishah	F	1
7	A'Jana	F	1
8	A'Janae	F	1
9	A'Lmos	M	1
10	A'Nette	F	1
11	A-Jay	M	1
12	A.	M	1
13	A.J.	M	2
14	Aaban	M	115
15	Aabha	F	35
16	Aabid	M	20
17	Aabidah	F	5
18	Aabigail	F	1
19	Aabir	M	10

```
# get frequencies to use later

namegraph_y = namegraph.reset_index().pivot(index='Name', columns='Gender', values='Count')
namegraph_y = namegraph_y.fillna(0)
namegraph_y['Fpercent'] = ((namegraph_y['F'] - namegraph_y['M'])/(namegraph_y['F'] + namegraph_y['M']))
namegraph_y['gender'] = np.where(namegraph_y['Fpercent'] > 0.001, 'female', 'male')
```

```
namegraph_y.head(20)
```

Gender Name	F	M	Fpercent	gender
A	2.0	2.0	0.0	male
A'Aff	1.0	0.0	1.0	female
A'Aron	0.0	1.0	-1.0	male
A'Dele	1.0	0.0	1.0	female
A'Isha	1.0	0.0	1.0	female
A'Ishah	1.0	0.0	1.0	female
A'Jana	1.0	0.0	1.0	female
A'Janae	1.0	0.0	1.0	female
A'Lmos	0.0	1.0	-1.0	male
A'Nette	1.0	0.0	1.0	female
A-Jay	0.0	1.0	-1.0	male
A.	0.0	1.0	-1.0	male
A.J.	0.0	2.0	-1.0	male
Aaban	0.0	115.0	-1.0	male
Aabha	35.0	0.0	1.0	female
Aabid	0.0	20.0	-1.0	male
Aabidah	5.0	0.0	1.0	female
Aabigail	1.0	0.0	1.0	female
Aabir	0.0	10.0	-1.0	male
Aabira	1.0	0.0	1.0	female

```
# train test split
char_vectorizer = CountVectorizer(analyzer='char',ngram_range=(2,2))
X = char_vectorizer.fit_transform(namegraph_y.index).tocsc()
y = (namegraph_y.gender == 'female').values


train, test = train_test_split(range(namegraph_y.shape[0]), train_size = 0.8)
mask=np.ones(namegraph_y.shape[0], dtype='int')
mask[train]=1
mask[test]=0
mask = (mask==1)


# show distribution graph (binary) of original
df_y = pd.DataFrame(y, columns=['Gender'])
g = sb.catplot(x="Gender", kind="count", data=df_y)
g.set_xticklabels(["Male", "Female"])

# where True is Female, False is Male
```

```
<seaborn.axisgrid.FacetGrid at 0x7f25476ca280>
```

This dataset is comprised of popular baby names derived from counts of a given name throughout various time periods. The dataset includes the attributes Name, Gender, Count, and Probability. The probability is calculated from the aggregate count. This model should predict whether a baby name is Male or Female. For our purposes, we created a new database that assigns a name male or female based off of percentage out of the aggregate.

Naïve Bayes


```
# no need for text preprocessing

# train
X_train = X[mask]
y_train = y[mask]
X_test = X[~mask]
y_test = y[~mask]

# do NB
naive_bayes = MultinomialNB(alpha = 1.0, class_prior=None, fit_prior=True)
naive_bayes.fit(X_train, y_train)

# output accuracy
train_accuracy = naive_bayes.score(X_train, y_train)
test_accuracy = naive_bayes.score(X_test, y_test)

print("Training accuracy:", train_accuracy)
print("Test accuracy:", test_accuracy)
```

```
Training accuracy: 0.7254779329400344
Test accuracy: 0.7228362332910163
```

```
# what the algorithm learned
# priors
prior_p = sum(y_train == 1)/len(y_train)
print('prior female:', prior_p, 'log of prior:', math.log(prior_p))
# output the prior the model learned
naive_bayes.class_log_prior_[1]
```

```
prior female: 0.6191285191546562 log of prior: -0.4794424040121086
-0.4794424040121079
```

```
# predict and evaluate
pred = naive_bayes.predict(X_test)

# print confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

```
array([[ 5901,  4286],
       [ 3137, 13458]])
```

```
# classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
                precision    recall  f1-score   support

         False       0.65      0.58      0.61     10187
          True       0.76      0.81      0.78     16595
```

```
          accuracy                           0.72     26782
         macro avg       0.71      0.70      0.70     26782
      weighted avg       0.72      0.72      0.72     26782
```

Logistic Regression

```
# train
classifier = LogisticRegression(solver='lbfgs', max_iter=250, class_weight='balanced')
classifier.fit(X_train, y_train)
```

```
LogisticRegression(class_weight='balanced', max_iter=250)
```

```
# evaluate
pred = classifier.predict(X_test)

# print confusion matrix
print(confusion_matrix(y_test, pred))

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, log_loss
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
probs = classifier.predict_proba(X_test)
print('log loss: ', log_loss(y_test, probs))
```

```
[[ 8149  2038]
 [ 3609 12986]]
accuracy score:  0.7891494287207826
precision score:  0.8643503727369543
recall score:  0.7825248568846038
f1 score:  0.8214048515133305
log loss:  0.4589416640515851
```

Neural Networks

```
# train
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier(solver='lbfgs', max_iter=500, alpha=1e-5,
                   hidden_layer_sizes=(15, 2), random_state=1)
classifier.fit(X_train, y_train)
```

```
pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.7701814651631693
precision score:  0.7993119266055045
recall score:  0.8400120518228382
f1 score:  0.8191567504039959
```

Analysis

Of all the machine-learning based approaches to text classification, I found that Logistic Regression scored the highest with a test accuracy of 0.79, Neural Networks came in with a close 0.77, and Naive Bayes with 0.72. Naive Bayes performs best with a small dataset. Given that this name-gender-dataset was relatively large with tens of thousands of entries, it makes sense that it was the poorest performing. However, all 3 algorithms performed fairly well. The original data was around 61% female names. Naive Bayes and Logistic Regression are suitable for binary classification, but Logistic Regression more so. This is evident as Logistic Regression had the best results. Neural Networks came in the

middle. All 3 are susceptible to overfitting and underfitting, but to mitigate this in Neural Networks we specify the number of layers and nodes using general guidelines. I did play around with different numbers but eventually decided to stick with the original from the example. The reason why is because Neural Network regression has a complex algorithm therefore it takes much longer to train. Overall, I found that they all did well. With further probing, we could analyze where the model was inaccurate and fix it to be closer in accuracy.