Text Classification

```
import pandas as pd
df = pd.read_csv('./CompiledjobListNigeria.csv', header=0, encoding='latin-1')
print('rows and columns:', df.shape)
print(df.head())
```

```
rows and columns: (202, 10)
                        ï»¿job_title             company_name  \
0                         Accountant     Equity Model Limited
1                     Content Writer        CLINTON FUND (CF)
2                         Accountant  Schleez Nigeria Limited
3                    Sales Executive  Bons Industries Limited
4  Bulk/Partnership Marketing Officer          TAMAK LOGISTICS

                       company_desc  \
0     Accounting, Auditing & Finance
1  Management & Business Development
2     Accounting, Auditing & Finance
3         Marketing & Communications
4         Marketing & Communications

                                          job_desc  \
0  Compiling, analyzing, and reporting financial ...
1  Creating, improving and maintaining content to...
2  Managing financial transactions, preparing fin...
3  Understanding of the sales process and dynamics."
4  Establish relationships with major businesses ...

                                    job_requirement            salary  \
0  This position is open preferably to a male can...  75,000 - 150,000
1   Bachelor's degree in Journalism, English, Com...  60,000 - 100,000
2  Minimum of Bachelor's degree in Accounting or ...        Negotiable
3  Minimum academic qualification of BSC/HND Degr...  75,000 - 150,000
4  Be smart & resourceful.,Great knowledge of how...  Less than 75,000

                                          location employment_type  \
0                                            Abuja       Full Time
1                                            Lagos       Full Time
2  First Floor, Left Wing, No. 49, Olowu Street, ...       Full-time
3                                            Enugu       Full Time
4                                            Lagos       Full Time

                    department  label
0             Law & Compliance      0
1              Content Writing      1
2                   Accounting      1
3  Manufacturing & Warehousing      0
4         Shipping & Logistics      0
```

The dataset consists of text information on Nigerian job listings. Some of these posts are real, and some of them are fake, and the goal is to create a binary classification predicting whether jobs are real or fake using the given fields. There are 9 features containing strings and 1 target column containing a numeric label.

```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

stopwords = (stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=stopwords)


# set up X and y
X = df.assign(concat = df['ï»¿job_title'].astype(str) + "  " + df.company_name.astype(str) + " " + df.company_desc.astype(str) + "  " + df.jo
           "  " + df.job_requirement.astype(str)+ "  " + df.location.astype(str) + " " + df.department.astype(str) + " " + df.employment_typ
X = X.concat
y = df.label

print(df[df['label'] == 1].count())
```

```
ï»¿job_title       67
company_name       67
company_desc       67
job_desc           67
job_requirement    67
salary             67
```

```
location           67
employment_type    67
department         67
label              67
dtype: int64
```

I chose to use 8 of the fields, as the salary one was numeric and seemed less relevant and more difficult to work with. Instead of combining vectorizers across weighted classes, I decided to simply concatenate all of the string fields into 1 column that can be preprocessed and vectorized. This method worked well with the Naive Bayes model, but the logistic regression model suffered on recall.

Text to Data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# apply tfidf vectorizer
X_train = vectorizer.fit_transform(X_train)  # transform only the train data

X_test = vectorizer.transform(X_test)        # transform only the test data

print(X_train.shape)
print(y_train.shape)
```

```
(161, 1867)
(161,)
```

Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

nb = MultinomialNB()
nb.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# make predictions on the test data
pred = nb.predict(X_test)

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

print('accuracy score: ', accuracy_score(y_test, pred))

print('\nprecision score (real): ', precision_score(y_test, pred, pos_label=0))
print('precision score (fake): ', precision_score(y_test, pred))

print('\nrecall score: (real)', recall_score(y_test, pred, pos_label=0))
print('recall score: (fake)', recall_score(y_test, pred))

print('\nf1 score: ', f1_score(y_test, pred))
```

```
[[27  0]
 [ 2 12]]
accuracy score:  0.9512195121951219

precision score (real):  0.9310344827586207
precision score (fake):  1.0

recall score: (real) 1.0
recall score: (fake) 0.8571428571428571

f1 score:  0.923076923076923
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import cohen_kappa_score
```

```
print("Kappa:", cohen_kappa_score(y_test, pred))
print(classification_report(y_test, pred))
```

```
Kappa: 0.8876712328767123
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        27
           1       1.00      0.86      0.92        14

    accuracy                           0.95        41
   macro avg       0.97      0.93      0.94        41
weighted avg       0.95      0.95      0.95        41
```

With only 202 instances, this data set is perfect for Naive Bayes, which historically performs well with small data sets. It converges quickly and has high bias, so it is prone to underfit data. The area in which this model struggled the most was recall on fake job posts; part of this is that there is a class imbalance towards real job posts, so given the option the model is more likely to classify an instance as real.

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver='lbfgs', random_state=1234)
lr.fit(X_train, y_train)
```

```
LogisticRegression(random_state=1234)
```

```
# make predictions on the test data
pred_log = lr.predict(X_test)

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

print('accuracy score: ', accuracy_score(y_test, pred_log))

print('\nprecision score (real): ', precision_score(y_test, pred_log, pos_label=0))
print('precision score (fake): ', precision_score(y_test, pred_log))

print('\nrecall score: (real)', recall_score(y_test, pred_log, pos_label=0))
print('recall score: (fake)', recall_score(y_test, pred_log))

print('\nf1 score: ', f1_score(y_test, pred_log))
```

```
[[27  0]
 [ 6  8]]
accuracy score:  0.8536585365853658

precision score (real):  0.8181818181818182
precision score (fake):  1.0

recall score: (real) 1.0
recall score: (fake) 0.5714285714285714

f1 score:  0.7272727272727273
```

```
print("Kappa:", cohen_kappa_score(y_test, pred_log))
print(classification_report(y_test, pred_log))
```

```
Kappa: 0.6371681415929203
              precision    recall  f1-score   support

           0       0.82      1.00      0.90        27
           1       1.00      0.57      0.73        14

    accuracy                           0.85        41
   macro avg       0.91      0.79      0.81        41
weighted avg       0.88      0.85      0.84        41
```

Logistic regression performed the worst out of all of the models. Logistic regression is a high-variance algorithm that converges more slowly than Naive Bayes, so it requires more instances in the training set. The poor 0.57 recall can be attributed to class imbalance. I verified this by including the Kappa metric, which shows that the accuracy adjusted for chance is much worse than what the accuracy value would suggest.

Neural Network

```
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(15, 3), random_state=1234)
classifier.fit(X_train, y_train)
```

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15, 3), random_state=1234,
              solver='lbfgs')
```

```
pred_nn = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred_nn))

print('\nprecision score (real): ', precision_score(y_test, pred_nn, pos_label=0))
print('precision score (fake): ', precision_score(y_test, pred_nn))

print('\nrecall score: (real)', recall_score(y_test, pred_nn, pos_label=0))
print('recall score: (fake)', recall_score(y_test, pred_nn))

print('f1 score: ', f1_score(y_test, pred_nn))
```

```
accuracy score:  0.975609756097561

precision score (real):  1.0
precision score (fake):  0.9333333333333333

recall score: (real) 0.9629629629629629
recall score: (fake) 1.0
f1 score:  0.9655172413793104
```

```
print("Kappa:", cohen_kappa_score(y_test, pred_nn))
print(classification_report(y_test, pred_nn))
```

```
Kappa: 0.9466840052015605
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        27
           1       0.93      1.00      0.97        14

    accuracy                           0.98        41
   macro avg       0.97      0.98      0.97        41
weighted avg       0.98      0.98      0.98        41
```

As expected, the neural network performed the best out of all of the algorithms. Neural networks are able to learn quickly and are adaptable to many situations, only requiring a change in architecture. I toyed around with several different hidden layer sizes and amounts, and through trial and error I found 15 and 3 nodes to work the best, which is coincidentally very similar to the example neural network on Github. Even when adjusting for random chance with the Kappa score, the neural network still performed admirably.

✓ 0s completed at 1:43 PM