Fake_News_Analysis_Modeling

November 3, 2025

[77]:
```
import pandas as pd
import numpy as np
import re



import string
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize, WhitespaceTokenizer
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression



from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score, f1_score, precision_score, recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
```

[54]:
```
# Load data
Buzzfeed = pd.read_csv('data/Buzzfeed_data.csv')
Buzzfeed_title = Buzzfeed.copy()
Buzzfeed_body = Buzzfeed.copy()

top1 = pd.read_csv('data/top1_fake_title.csv').head(5)  # top 5 fake title words
top2 = pd.read_csv('data/top2_real_title.csv').head(5)  # top 5 real title words
top3 = pd.read_csv('data/top3_fake_body.csv').head(5)   # top 5 fake body words
top4 = pd.read_csv('data/top4_real_body.csv').head(5)   # top 5 real body words
```

[56]:
```
# Convert top words to list
fake_title_words = top1['word'].tolist()
real_title_words = top2['word'].tolist()
fake_body_words = top3['word'].tolist()
real_body_words = top4['word'].tolist()


# Helper to count occurrences of a word
# def count_word(text, word):
#     if isinstance(text, str):
#         return len(re.findall(rf'\b{word}\b', text))
#     else:
#         return 0

def has_word(text, word):
    if isinstance(text, str):
        return 1 if re.search(rf'\b{word}\b', text) else 0
    else:
        return 0
```

[57]:
```
# # Create fake title word columns
# for word in fake_title_words:
#     col_name = f"fake_title_{word}"
#     Buzzfeed_title[col_name] = Buzzfeed_title['title'].apply(lambda x:␣
 ↪count_word(x, word))

# # Create real title word columns
# for word in real_title_words:
#     col_name = f"real_title_{word}"
#     Buzzfeed_title[col_name] = Buzzfeed_title['title'].apply(lambda x:␣
 ↪count_word(x, word))


# # Create fake body word columns
# for word in fake_body_words:
#     col_name = f"fake_body_{word}"
#     Buzzfeed_body[col_name] = Buzzfeed_body['text'].apply(lambda x:␣
 ↪count_word(x, word))

# # Create real body word columns
# for word in real_body_words:
#     col_name = f"real_body_{word}"
#     Buzzfeed_body[col_name] = Buzzfeed_body['text'].apply(lambda x:␣
 ↪count_word(x, word))

# Create fake title word columns
for word in fake_title_words:
```

```
    col_name = f"fake_title_{word}"
    Buzzfeed_title[col_name] = Buzzfeed_title['title'].apply(lambda x:␣
 ↪has_word(x, word))

# Create real title word columns
for word in real_title_words:
    col_name = f"real_title_{word}"
    Buzzfeed_title[col_name] = Buzzfeed_title['title'].apply(lambda x:␣
 ↪has_word(x, word))


# Create fake body word columns
for word in fake_body_words:
    col_name = f"fake_body_{word}"
    Buzzfeed_body[col_name] = Buzzfeed_body['text'].apply(lambda x: has_word(x,␣
 ↪word))

# Create real body word columns
for word in real_body_words:
    col_name = f"real_body_{word}"
    Buzzfeed_body[col_name] = Buzzfeed_body['text'].apply(lambda x: has_word(x,␣
 ↪word))
```

[66]:
```
# Show new columns
# Start from your Buzzfeed_title DataFrame
title = Buzzfeed_title.copy()

# Drop columns that shouldn't be used as predictors
title = title.drop(columns=['title', 'text', 'source'])

# # Convert 'source' to categorical (one-hot encoding)
# title = pd.get_dummies(title, columns=['source'], drop_first=True)

# Encode the target variable (news_type)
le = LabelEncoder()
title['news_type'] = le.fit_transform(title['news_type'])

title.head(3)
```

[66]:
```
   news_type  contain_movies  contain_images  fake_title_hillari  \
0          1               0               1                   0
1          1               0               1                   0
2          1               1               1                   0

   fake_title_clinton  fake_title_obama  fake_title_freedom  fake_title_daili  \
0                   0                 0                   0                 0
1                   0                 0                   0                 0
```

```
2                   0                 0                   0                 0

   real_title_trump  real_title_clinton  real_title_donald  real_title_debat  \
0                 0                   0                  0                 0
1                 0                   0                  0                 0
2                 0                   0                  0                 0

   real_title_obama
0                 0
1                 0
2                 0
```

[68]:
```
# Show new columns
# Start from your Buzzfeed_body DataFrame
body = Buzzfeed_body.copy()

# Drop columns that shouldn't be used as predictors
body = body.drop(columns=['title', 'text','source'])

# # Convert 'source' to categorical (one-hot encoding)
# body = pd.get_dummies(body, columns=['source'], drop_first=True)

# Encode the target variable (news_type)
le = LabelEncoder()
body['news_type'] = le.fit_transform(body['news_type'])

body.head(3)
```

[68]:
```
   news_type  contain_movies  contain_images  fake_body_clinton  \
0          1               0               1                  0
1          1               0               1                  0
2          1               1               1                  0

   fake_body_hillari  fake_body_trump  fake_body_peopl  fake_body_just  \
0                  0                0                0               1
1                  0                0                0               0
2                  0                0                0               1

   real_body_trump  real_body_said  real_body_clinton  real_body_say  \
0                0               1                  0              1
1                0               1                  0              1
2                0               1                  0              1

   real_body_debat
0                0
1                0
2                0
```

[69]:
```
# Define features and target
X = title
y = title['news_type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Logistic Regression model
lr = LogisticRegression(max_iter=500)

# Forward Selection (choose 10 best features)
sfs_forward = SFS(lr,
                  k_features=10,
                  forward=True,
                  floating=False,
                  scoring='accuracy',
                  cv=5)

sfs_forward = sfs_forward.fit(X_train, y_train)

print("Selected Features (Forward):")
print(list(sfs_forward.k_feature_names_))
```

```
Selected Features (Forward):
['news_type', 'contain_movies', 'contain_images', 'fake_title_hillari',
'fake_title_clinton', 'fake_title_obama', 'fake_title_freedom',
'fake_title_daili', 'real_title_trump', 'real_title_clinton']
```

[70]:
```
# Full Model (all features)
lr_full = LogisticRegression(max_iter=1000, solver='liblinear')
lr_full.fit(X_train, y_train)

y_pred_full = lr_full.predict(X_test)
acc_full = accuracy_score(y_test, y_pred_full)
print(f"Full Model Accuracy: {acc_full:.4f}")

# Selected Feature Model (using SFS)
selected_features = list(sfs_forward.k_feature_names_)
X_train_sfs = X_train[selected_features]
X_test_sfs = X_test[selected_features]

lr_sfs = LogisticRegression(max_iter=1000, solver='liblinear')
lr_sfs.fit(X_train_sfs, y_train)

y_pred_sfs = lr_sfs.predict(X_test_sfs)
acc_sfs = accuracy_score(y_test, y_pred_sfs)
print(f"Selected Feature Model Accuracy: {acc_sfs:.4f}")
```

```
# Comparison
improvement = acc_sfs - acc_full
print(f"Accuracy Difference (SFS - Full): {improvement:.4f}")
```

```
Full Model Accuracy: 1.0000
Selected Feature Model Accuracy: 1.0000
Accuracy Difference (SFS - Full): 0.0000
```

[ ]:
```
# Check class balance
print(title['news_type'].value_counts(normalize=True))

# Check duplicates / leakage
print(Buzzfeed.duplicated(subset=['title','text']).sum())
```

```
0    0.5
1    0.5
Name: news_type, dtype: float64
4
```

[41]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression
import numpy as np

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
lr = LogisticRegression(max_iter=2000, solver='liblinear')

scores = cross_val_score(lr, X, y, cv=cv, scoring='accuracy', n_jobs=-1)
print("CV accuracies:", scores)
print("Mean CV accuracy: {:.3f} ± {:.3f}".format(scores.mean(), scores.std()))
```

```
CV accuracies: [1. 1. 1. 1. 1.]
Mean CV accuracy: 1.000 ± 0.000
```

[ ]:
```
# Text Preprocessing Functions

ps = PorterStemmer()
wst = WhitespaceTokenizer()

# Lowercase
def lower_func(x):
    return x.lower()

# Remove numbers
def remove_number_func(x):
    return ''.join([a for a in x if not a.isdigit()])

# Remove punctuation
def remove_punc_func(x):
```

```
    return ''.join([a for a in x if a not in string.punctuation])

# Remove special characters
def remove_spec_char_func(x):
    return ''.join([a for a in x if a.isalnum() or a == ' '])

# Remove English stopwords (using sklearn)
def remove_stopwords(x):
    new = []
    for a in x.split():
        if a not in ENGLISH_STOP_WORDS:
            new.append(a)
    return " ".join(new)

# Stemming
def stem_func(x):
    wordlist = word_tokenize(x)
    psstem = [ps.stem(a) for a in wordlist]
    return ' '.join(psstem)

# Remove extra whitespaces
def remove_whitespace_func(x):
    return(wst.tokenize(x))

# Function composition helper
def compose(f, g):
    return lambda x: f(g(x))

# Final preprocessing pipeline
final = compose(
    compose(
        compose(
            compose(
                compose(
                    compose(remove_whitespace_func, stem_func),
                    remove_stopwords
                ),
                remove_spec_char_func
            ),
            remove_punc_func
        ),
        remove_number_func
    ),
    lower_func
)
```

[74]:
```
# Split features and target
X = Buzzfeed['title']
y = Buzzfeed['news_type']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42, stratify=y)

# Preprocessing + RandomForest pipeline
pp = Pipeline([
    ('bow', CountVectorizer(analyzer=final)),  # final is your preprocessing␣
 ↪function
    ('tfidf', TfidfTransformer()),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Fit model
pp.fit(X_train, y_train)

# Predictions
predictions = pp.predict(X_test)

# Evaluate
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
[[13 15]
 [ 3 24]]
              precision    recall  f1-score   support

        Fake       0.81      0.46      0.59        28
        Real       0.62      0.89      0.73        27

    accuracy                           0.67        55
   macro avg       0.71      0.68      0.66        55
weighted avg       0.72      0.67      0.66        55
```

[75]:
```
# Split features and target
X = Buzzfeed['text']
y = Buzzfeed['news_type']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42, stratify=y)

# Preprocessing + RandomForest pipeline
```

```
pp = Pipeline([
    ('bow', CountVectorizer(analyzer=final)),  # final is your preprocessing␣
 ↪function
    ('tfidf', TfidfTransformer()),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Fit model
pp.fit(X_train, y_train)

# Predictions
predictions = pp.predict(X_test)

# Evaluate
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
[[19  9]
 [ 8 19]]
              precision    recall  f1-score   support

        Fake       0.70      0.68      0.69        28
        Real       0.68      0.70      0.69        27

    accuracy                           0.69        55
   macro avg       0.69      0.69      0.69        55
weighted avg       0.69      0.69      0.69        55
```

[ ]:
```
# Define models to compare
models = {
    "RandomForest": RandomForestClassifier(n_estimators=100, random_state=42),
    "LogisticRegression": LogisticRegression(max_iter=500, random_state=42),
    "NaiveBayes": MultinomialNB(),
    "SVM": SVC(kernel='linear', random_state=42)
}

# Feature types
feature_types = {
    "Title": Buzzfeed['title'],
    "Body": Buzzfeed['text']
}

results = []

for feat_name, X in feature_types.items():
    y = Buzzfeed['news_type']
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42, stratify=y)

    for model_name, model in models.items():
        pipeline = Pipeline([
            ('bow', CountVectorizer(analyzer=final)),
            ('tfidf', TfidfTransformer()),
            ('clf', model)
        ])

        pipeline.fit(X_train, y_train)
        preds = pipeline.predict(X_test)

        results.append({
            "Feature": feat_name,
            "Model": model_name,
            "Accuracy": accuracy_score(y_test, preds),
            "Precision": precision_score(y_test, preds, pos_label='Real'),
            "Recall": recall_score(y_test, preds, pos_label='Real'),
            "F1": f1_score(y_test, preds, pos_label='Real')
        })

# Convert results to DataFrame
results_df = pd.DataFrame(results)

# Pivot table to show only accuracy
accuracy_df = results_df.pivot(index='Model', columns='Feature',␣
↪values='Accuracy')
accuracy_df = accuracy_df.reset_index()
accuracy_df
```

	Feature	Model	Accuracy	Precision	Recall	F1
0	Title	RandomForest	0.672727	0.615385	0.888889	0.727273
1	Title	LogisticRegression	0.690909	0.656250	0.777778	0.711864
2	Title	NaiveBayes	0.618182	0.607143	0.629630	0.618182
3	Title	SVM	0.654545	0.633333	0.703704	0.666667
4	Body	RandomForest	0.690909	0.678571	0.703704	0.690909
5	Body	LogisticRegression	0.781818	0.714286	0.925926	0.806452
6	Body	NaiveBayes	0.763636	0.694444	0.925926	0.793651
7	Body	SVM	0.818182	0.757576	0.925926	0.833333

[81]:
```
# Pivot table to show only accuracy
accuracy_df = results_df.pivot(index='Model', columns='Feature',␣
↪values='Accuracy')
accuracy_df = accuracy_df.reset_index()
accuracy_df
```

[81]:
Feature	Model	Body	Title
0	LogisticRegression	0.781818	0.690909
1	NaiveBayes	0.763636	0.618182
2	RandomForest	0.690909	0.672727
3	SVM	0.818182	0.654545

[84]:
```
import joblib
import os

def finals(text):
    text = lower_func(text)
    text = remove_number_func(text)
    text = remove_punc_func(text)
    text = remove_spec_char_func(text)
    text = remove_stopwords(text)
    text = stem_func(text)
    text = ' '.join(remove_whitespace_func(text))
    return text

# Split features and target
X = Buzzfeed['text']
y = Buzzfeed['news_type']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Preprocessing + SVM pipeline
svm_pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=finals)),  # your preprocessing function
    ('tfidf', TfidfTransformer()),
    ('classifier', SVC(kernel='linear', random_state=42))
])

# Fit model
svm_pipeline.fit(X_train, y_train)


# Path to folder
os.makedirs('data', exist_ok=True)

# Save your trained pipeline inside the data folder
joblib.dump(svm_pipeline, 'data/svm_body_model.pkl')
```

[84]:
```
['data/svm_body_model.pkl']
```