PA3b: Sentiment classification

Tim Boleslawsky & Mikolaj Nowacki

In [1]:
```python
# Importing libraries and setting globals
import pandas as pd
import numpy as np
import re
from rapidfuzz import process, fuzz
from sklearn.metrics import cohen_kappa_score
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from imblearn.combine import SMOTEENN
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, classification_report
import time
RANDOM_SEED = 0
```

In [2]:
```python
# Reading the CSV files with error handling
crowdsourced_train = pd.read_csv('crowdsourced_train.csv', sep='\t')
gold_train = pd.read_csv('gold_train.csv', sep='\t')
test = pd.read_csv('test.csv', sep='\t')
```

Data Munging and Exploratory Data Analysis

Data Cleaning - Ensuring data compatibility by handling missing values and errors

In [3]:
```python
# Initial Analysis
# Check unique values
print(crowdsourced_train["sentiment"].unique())
print(gold_train["sentiment"].unique())
print(test["sentiment"].unique())

# Look for missing values
print(crowdsourced_train["sentiment"].isnull().sum())
print(gold_train["sentiment"].isnull().sum())
print(test["sentiment"].isnull().sum())
```

```
['Positive' 'Negative' 'Neutral' 'neutral' 'negative' 'positive'
 'neutral?' 'Nuetral' 'negative ' 'positive ' 'neutral ' ' negative'
 'Nedative' '_x0008_neutral' 'netural' 'netutral' 'Neutral ' 'Neural'
 'Neutrall' 'Netural' 'neugral' 'neutrla' 'negtaive' 'postive'
 '                neutral'
 '                                        positive' '    positive'
 ' neutral' '               negative' 'Nutral' 'negayive' 'positie'
 'neutra l' 'npositive' 'neutal' 'postitive' 'Positve']
['neutral' 'positive' 'negative']
['neutral' 'negative' 'positive']
0
0
0
```

In [4]:
```python
# Define sentiment cleaning function
def clean_sentiment_fuzzy(label):
    valid_labels = ['positive', 'negative', 'neutral']

    # Normalize string and remove punctuation, etc.
    label = str(label).lower().strip()
    label = re.sub(r'[^a-z]', '', label)

    # Use fuzzy matching to find closest valid label
    match, score, _ = process.extractOne(label, valid_labels, scorer=fuzz.ratio)

    if score >= 70:
        return match
    else:
        return 'unknown'
```

In [5]:
```python
# Apply cleaning function
crowdsourced_train['cleaned_sentiment'] = crowdsourced_train['sentiment'].apply(clean_sentiment_fuzzy)
crowdsourced_train["cleaned_sentiment"].unique()
```

Out[5]:
```
array(['positive', 'negative', 'neutral'], dtype=object)
```

In [6]:
```python
# Inter-Annotator Agreement
kappa = cohen_kappa_score(crowdsourced_train["sentiment"], gold_train["sentiment"])
print(kappa)
```

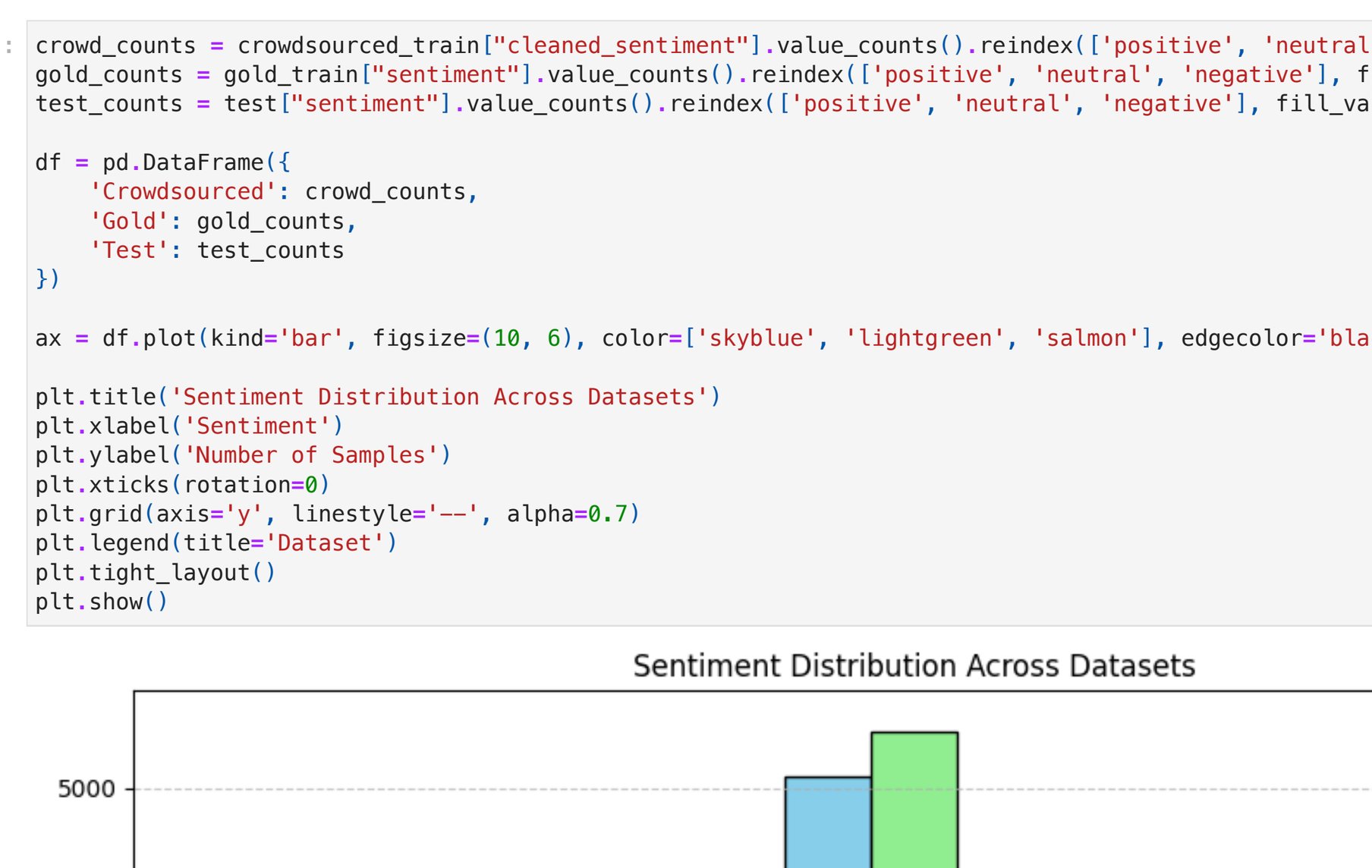
```
0.1933310606962546
```

Exploratory Data Analysis

In [7]:
```python
# EDA for crowdsourced data
sentiment_counts = crowdsourced_train["cleaned_sentiment"].value_counts()
sentiment_counts = sentiment_counts.reindex(['positive', 'neutral', 'negative'])

plt.figure(figsize=(8, 5))
sentiment_counts.plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Number of Samples')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

In [8]:
```python
# EDA for gold data
sentiment_counts = gold_train["sentiment"].value_counts()
sentiment_counts = sentiment_counts.reindex(['positive', 'neutral', 'negative'])

plt.figure(figsize=(8, 5))
sentiment_counts.plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Number of Samples')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

In [9]:
```python
# EDA for test data
sentiment_counts = test["sentiment"].value_counts()
sentiment_counts = sentiment_counts.reindex(['positive', 'neutral', 'negative'])

plt.figure(figsize=(8, 5))
sentiment_counts.plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Number of Samples')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



In [10]:
```python
crowd_counts = crowdsourced_train["cleaned_sentiment"].value_counts().reindex(['positive', 'neutral', 'negative'], fill_value=0)
gold_counts = gold_train["sentiment"].value_counts().reindex(['positive', 'neutral', 'negative'], fill_value=0)
test_counts = test["sentiment"].value_counts().reindex(['positive', 'neutral', 'negative'], fill_value=0)

df = pd.DataFrame({
    'Crowdsourced': crowd_counts,
    'Gold': gold_counts,
    'Test': test_counts
})

ax = df.plot(kind='bar', figsize=(10, 6), color=['skyblue', 'lightgreen', 'salmon'], edgecolor='black')

plt.title('Sentiment Distribution Across Datasets')
plt.xlabel('Sentiment')
plt.ylabel('Number of Samples')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend(title='Dataset')
plt.tight_layout()
plt.show()
```

Data Preprocessing

In [11]:
```python
# Data Encoding
le = LabelEncoder()
le.fit(pd.concat([crowdsourced_train['cleaned_sentiment'], gold_train['sentiment']]))
crowdsourced_train['cleaned_sentiment_encoded'] = le.transform(crowdsourced_train['cleaned_sentiment'])
gold_train['sentiment_encoded'] = le.transform(gold_train['sentiment'])
test['sentiment_encoded'] = le.transform(test['sentiment'])
```

In [12]:
```python
# Defining variables
X_crowdsourced = crowdsourced_train['text']
Y_crowdsourced = crowdsourced_train['cleaned_sentiment_encoded']

X_gold = gold_train['text']
Y_gold = gold_train['sentiment_encoded']

X_test = test['text']
Y_test = test['sentiment_encoded']
```

In [13]:
```python
# Transformation
vectorizer = TfidfVectorizer()
vectorizer.fit(X_crowdsourced + X_gold)
X_crowdsourced_vec = vectorizer.transform(X_crowdsourced)
X_gold_vec = vectorizer.transform(X_gold)
X_test_vec = vectorizer.transform(X_test)
```

Building the Model

In [14]:
```python
# Baseline model
clf = DummyClassifier(strategy='most_frequent')

dummy_val_score = cross_val_score(clf, X_crowdsourced_vec, Y_crowdsourced)
dummy_mean_score = np.mean(dummy_val_score)
print(dummy_mean_score)
```

```
0.475739985439746
```

In [15]:
```python
# Logistic Regression
log_reg_clf = LogisticRegression(max_iter=1000, random_state=RANDOM_SEED, class_weight='balanced')
log_reg_score = cross_val_score(log_reg_clf, X_crowdsourced_vec, Y_crowdsourced)
log_reg_mean = np.mean(log_reg_score)

# Multinomial Naive Bayes
nb_clf = MultinomialNB()
nb_score = cross_val_score(nb_clf, X_crowdsourced_vec, Y_crowdsourced)
nb_mean = np.mean(nb_score)

# LinearSVC
svc_clf = LinearSVC(random_state=RANDOM_SEED, class_weight='balanced')
svc_score = cross_val_score(svc_clf, X_crowdsourced_vec, Y_crowdsourced)
svc_mean = np.mean(svc_score)

print("Logistic Regression average accuracy over 5 folds: ", log_reg_mean)
print("Multinomial Naive Bayes average accuracy over 5 folds: ", nb_mean)
print("LinearSVC average accuracy over 5 folds: ", svc_mean)
```

```
Logistic Regression average accuracy over 5 folds:  0.5645370540922208
Multinomial Naive Bayes average accuracy over 5 folds:  0.5066505275899271
LinearSVC average accuracy over 5 folds:  0.5578861756527993
```

In [16]:
```python
subset_sizes = [0.1, 0.25, 0.5, 0.75]
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=RANDOM_SEED),
    "Multinomial Naive Bayes": MultinomialNB(),
    "Linear SVC": LinearSVC(random_state=RANDOM_SEED)
}

for name, model in classifiers.items():
    print(f"\n{name}")
    for frac in subset_sizes:
        X_sub, _, y_sub, _ = train_test_split(X_crowdsourced_vec, Y_crowdsourced, train_size=frac, random_state=RANDOM_SEED)

        start = time.time()
        model.fit(X_sub, y_sub)
        train_time = time.time() - start

        print(f"  Size: {int(frac * 100)}% | Train time: {train_time:.3f}s")
```

```
Logistic Regression
  Size: 10% | Train time: 2.490s
  Size: 25% | Train time: 5.498s
  Size: 50% | Train time: 5.968s
  Size: 75% | Train time: 6.480s

Multinomial Naive Bayes
  Size: 10% | Train time: 0.002s
  Size: 25% | Train time: 0.002s
  Size: 50% | Train time: 0.002s
  Size: 75% | Train time: 0.004s

Linear SVC
  Size: 10% | Train time: 0.008s
  Size: 25% | Train time: 0.020s
  Size: 50% | Train time: 0.053s
  Size: 75% | Train time: 0.081s
```

In [17]:
```python
param_grid = {
    'C': [0.01, 0.1, 0.5, 1, 5, 10],
    'dual': [True, False],
    'max_iter': [1000, 2000, 3000],
}

svc = LinearSVC(random_state=RANDOM_SEED)
search = GridSearchCV(svc, param_grid, cv=5, scoring='accuracy', error_score='raise')
search.fit(X_crowdsourced_vec, Y_crowdsourced)

print("Best parameters:", search.best_params_)
print("Best cross-val accuracy:", search.best_score_)
```

```
Best parameters: {'C': 0.5, 'dual': False, 'max_iter': 1000}
Best cross-val accuracy: 0.573622433316668
```

Testing the Model

In [18]:
```python
# Testing the model on the test set with the tuneded parameters
tuned_svc = LinearSVC(random_state=RANDOM_SEED, C=0.5, dual=False, max_iter=1000)
tuned_svc.fit(X_crowdsourced_vec, Y_crowdsourced)

pred = tuned_svc.predict(X_test_vec)
accuracy = accuracy_score(Y_test, pred)
f1 = f1_score(Y_test, pred, average='weighted')
print("Accuracy on test set with crowdsource data:", accuracy)
print("F1 score on test set with crowdsource data:", f1)
```

```
Accuracy on test set with crowdsource data: 0.6203837798696596
F1 score on test set with crowdsource data: 0.6184433629992475
```

In [19]:
```python
report = classification_report(Y_test, pred, target_names=['negative', 'neutral', 'positive'])
print(report)
```

```
              precision    recall  f1-score   support

    negative       0.54      0.54      0.54      1077
     neutral       0.62      0.71      0.66      2597
    positive       0.68      0.54      0.60      1850

    accuracy                           0.62      5524
   macro avg       0.61      0.60      0.60      5524
weighted avg       0.63      0.62      0.62      5524
```

In [20]:
```python
# Investigating feature importance per class
feature_names = vectorizer.get_feature_names_out()

coefs = tuned_svc.coef_

for i, class_label in enumerate(tuned_svc.classes_):
    print(f"\nTop positive features for class '{class_label}':")

    # Sort coefficients by importance
    top_positive_indices = np.argsort(coefs[i])[-10:]
    top_negative_indices = np.argsort(coefs[i])[:10]

    print("Top positively correlated words:")
    print(pd.DataFrame({
        'Feature': feature_names[top_positive_indices],
        'Coefficient': coefs[i][top_positive_indices]
    }).sort_values(by='Coefficient', ascending=False))

    print("\nTop negatively correlated words:")
    print(pd.DataFrame({
        'Feature': feature_names[top_negative_indices],
        'Coefficient': coefs[i][top_negative_indices]
    }).sort_values(by='Coefficient'))
```

```
Top positive features for class '0':
Top positively correlated words:
   Feature  Coefficient
9    worst     2.601372
8      sad     2.379596
7     fuck     1.937336
6    upset     1.884734
5     hate     1.837314
4   stupid     1.776412
3     ugly     1.612434
2  muslims     1.608250
1    yakub     1.594090
0     shit     1.556096

Top negatively correlated words:
           Feature  Coefficient
0             love    -1.325569
1            great    -1.130226
2          amazing    -1.101215
3             june    -1.066672
4             thor    -1.063598
5  govmikehuckabee    -1.046948
6            happy    -1.037120
7              hot    -1.019354
8              ant    -1.016885
9          nirvana    -0.987539

Top positive features for class '1':
Top positively correlated words:
    Feature  Coefficient
9     gucci     1.311619
8        or     1.293229
7     asked     1.259952
6      liam     1.167999
5    around     1.167698
4    curtis     1.132620
3      grew     1.122374
2    finish     1.102638
1    ignore     1.097491
0  entrance     1.094836

Top negatively correlated words:
   Feature  Coefficient
0    happy    -2.730545
1  excited    -1.926565
2  fucking    -1.822667
3     such    -1.689000
4    proud    -1.617073
5     best    -1.582857
6     love    -1.579233
7      wow    -1.572516
8     good    -1.538289
9     wait    -1.533075

Top positive features for class '2':
Top positively correlated words:
   Feature  Coefficient
9    happy     3.378092
8     love     2.549174
7  amazing     2.430340
6  excited     2.356986
5     best     2.258459
4     wait     2.142094
3    proud     2.082495
2  awesome     1.939948
1     good     1.866639
0  finally     1.755684

Top negatively correlated words:
   Feature  Coefficient
0    worst    -1.488411
1    louis    -1.325371
2    yakub    -1.220281
3   rather    -1.190320
4      ira    -1.120764
5     iran    -1.108675
6     tell    -1.098270
7      not    -1.068892
8       or    -1.020780
9  instead    -1.017096
```

In [25]:
```python
# Investigating false predictions
mistakes = np.where(Y_test != pred)[0]

for idx in np.random.choice(mistakes, size=5, replace=False):
    print(f"True label: {Y_test[idx]} --- Predicted: {pred[idx]}")
    print(f"Text: {X_crowdsourced[idx]}\n")
```

```
True label: 0 --- Predicted: 1
Text: @CNN So what If the rules were spelled out back in May.  Politics is all about change.  Carly Fiorina deserves a place in your debate now.

True label: 1 --- Predicted: 0
Text: It's Friday. 3 day weekends upon us. Won Red Sox tickets for tonight. And I get out of work early to go #Winning #BestFridayEver #MommyNMe

True label: 2 --- Predicted: 1
Text: Super Eagles play Tanzania today.   Most Nigerians are just waiting for them to lose so they lambast Sunday oliseh and the whole of NFF.

True label: 0 --- Predicted: 1
Text: I'm on the 5th episode of this season of Arrow, Curtis is a new favourite character, Curtis and Felicity are the only ones I really like

True label: 0 --- Predicted: 1
Text: 12 yrs old and "gay"...the new American heroes https://t.co/In4ICAOVH1
```

In [26]:
```python
# Testing the model on the test set with the tuneded parameters
tuned_svc = LinearSVC(random_state=RANDOM_SEED, C=0.5, dual=False, max_iter=1000)
tuned_svc.fit(X_gold_vec, Y_gold)

pred = tuned_svc.predict(X_test_vec)
accuracy = accuracy_score(Y_test, pred)
f1 = f1_score(Y_test, pred, average='weighted')
print("Accuracy on test set with gold data:", accuracy)
print("F1 score on test set with gold data:", f1)
```

```
Accuracy on test set with gold data: 0.780955829109341
F1 score on test set with gold data: 0.7791527011944493
```