Annexe A: Première partie de la compétition

1 Kaggle competition 1

1.1 I) Creating hypothesis for relationships within data, visualizing the data, understanding and describing the data

1.1.1 Par Emiliano Aviles et Cassandre Hamel

[6]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import TruncatedSVD
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, f1_score
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from tqdm import tqdm
import matplotlib.pyplot as plt
```

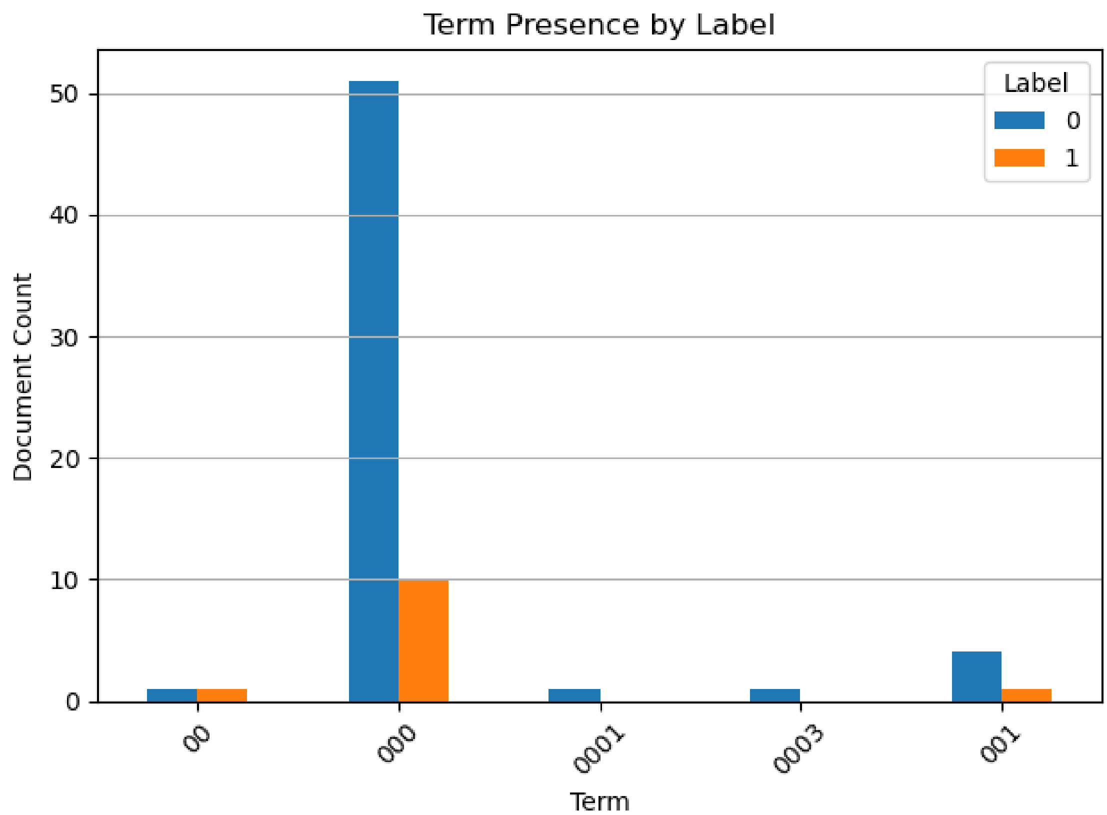
1.1.2 1) Creating a DataFrame of the Data, visualizing contingency tables

[7]:
```
# Load data
data_train = np.load('data_train.npy', allow_pickle = True)
data_test = np.load('data_test.npy', allow_pickle = True)
vocab_map = np.load('vocab_map.npy', allow_pickle=True)

# Load labels_train from CSV and extract the 'label' column
labels_train_df = pd.read_csv('label_train.csv')  # Assuming this is your labels␣
↪file
labels_train = labels_train_df['label'].values  # Extract the labels as a NumPy␣
↪array

# Convert training data to a DataFrame for visualization
df_train = pd.DataFrame(data_train)

# Add column names using vocab_map
df_train.columns = vocab_map

# Add the target labels to the DataFrame
df_train['TARGETT'] = labels_train

# df_train.to_csv("df_train.csv")
```

```python
# Select a few important terms for barplot analysis
selected_terms = vocab_map[:5]  # Let's analyze the first 5 terms

# Prepare data for barplot
term_frequencies = []
labels = []

# Select a few important terms for creating contingency tables
selected_terms = vocab_map[:5]  # Let's say we are analyzing the first 5 terms

# Create contingency tables for each selected term
for term in selected_terms:
    # Convert term counts to binary (presence/absence)
    df_train[term + '_present'] = df_train[term].apply(lambda x: 1 if x > 0 else␣
 ↪0)

    # Generate a contingency table for the term
    contingency_table = pd.crosstab(df_train[term + '_present'],␣
 ↪df_train['TARGETT'])

    print(f"Contingency table for term: {term}")
    print(contingency_table)
    print("\n")

    # Extract frequencies for term presence (row '1')
    if 1 in contingency_table.index:
        term_frequencies.append(contingency_table.loc[1])
        labels.append(term)

# Convert list to DataFrame for plotting
freq_df = pd.DataFrame(term_frequencies, index=labels).fillna(0)

# Plotting
plt.figure(figsize=(10, 6))
freq_df.plot(kind='bar', stacked=False)
plt.title('Term Presence by Label')
plt.xlabel('Term')
plt.ylabel('Document Count')
plt.xticks(rotation=45)
plt.legend(title='Label')
plt.grid(axis='y')
plt.tight_layout()
plt.show()

print('''
Interpretation:
```

```
Label 0: There are 7,123 documents where the term "00" is absent (not present)␣
 ↪and the label is 0.
Label 1: There are 2,297 documents where the term "00" is absent and the label␣
 ↪is 1.

Label 0: There is 1 document where the term "00" is present and the label is 0.
Label 1: There is 1 document where the term "00" is present and the label is 1.
''')
```

```
Contingency table for term: 00
TARGETT        0     1
00_present
0           7123  2297
1              1     1


Contingency table for term: 000
TARGETT         0     1
000_present
0            7073  2288
1              51    10


Contingency table for term: 0001
TARGETT          0     1
0001_present
0             7123  2298
1                1     0


Contingency table for term: 0003
TARGETT          0     1
0003_present
0             7123  2298
1                1     0


Contingency table for term: 001
TARGETT         0     1
001_present
0            7120  2297
1               4     1


<Figure size 1000x600 with 0 Axes>
```




```
Interpretation:

Label 0: There are 7,123 documents where the term "00" is absent (not present)
and the label is 0.
Label 1: There are 2,297 documents where the term "00" is absent and the label
is 1.

Label 0: There is 1 document where the term "00" is present and the label is 0.
Label 1: There is 1 document where the term "00" is present and the label is 1.
```

[4]:
```
df_train = df_train.iloc[:, :-1]
df_train.head()
```

[4]:

	00	000	0001	0003	001	01	0112	019	02	023	...	zooms	zoph	\
0	0	0	0	0	0	0	0	0	0	0	...	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	

```
   zoroastrian  zp  zpg  zps  zs  zsda  zsl  TARGETT
0            0   0    0    0   0     0    0        0
1            0   0    0    0   0     0    0        0
2            0   0    0    0   0     0    0        0
3            0   0    0    0   0     0    0        0
4            0   0    0    0   0     0    0        1

[5 rows x 26355 columns]
```

[8]:
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, f1_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

[10]:
```python
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.decomposition import TruncatedSVD
```

1.1.3 2) Transform the data with TF-IDF and visualize the DataFrame

[9]:
```python
tfidf_transformer = TfidfTransformer()
data_train_tfidf = tfidf_transformer.fit_transform(data_train)
data_test_tfidf = tfidf_transformer.transform(data_test)

# Convert the sparse matrix to a dense format (if needed)
data_train_tfidf_dense = data_train_tfidf.toarray()

# Create a DataFrame for easier visualization, with terms as column names
df_tfidf = pd.DataFrame(data_train_tfidf_dense, columns=vocab_map)
```

[10]:
```python
# Display the first few rows of the DataFrame
print(df_tfidf.shape)
print(df_tfidf.head())
```

```
(9422, 26354)
    00  000  0001  0003  001   01  0112  019   02  023  ...  zoom  zooms  \
0  0.0  0.0   0.0   0.0  0.0  0.0   0.0  0.0  0.0  0.0  ...   0.0    0.0
1  0.0  0.0   0.0   0.0  0.0  0.0   0.0  0.0  0.0  0.0  ...   0.0    0.0
2  0.0  0.0   0.0   0.0  0.0  0.0   0.0  0.0  0.0  0.0  ...   0.0    0.0
3  0.0  0.0   0.0   0.0  0.0  0.0   0.0  0.0  0.0  0.0  ...   0.0    0.0
4  0.0  0.0   0.0   0.0  0.0  0.0   0.0  0.0  0.0  0.0  ...   0.0    0.0
```

```
   zoph  zoroastrian   zp  zpg  zps   zs  zsda  zsl
0   0.0          0.0  0.0  0.0  0.0  0.0   0.0  0.0
1   0.0          0.0  0.0  0.0  0.0  0.0   0.0  0.0
2   0.0          0.0  0.0  0.0  0.0  0.0   0.0  0.0
3   0.0          0.0  0.0  0.0  0.0  0.0   0.0  0.0
4   0.0          0.0  0.0  0.0  0.0  0.0   0.0  0.0

[5 rows x 26354 columns]
```

1.2 II) Dimensionalty reduction, and model selection

1.2.1 1) Dimensionality reduction using truncated SVD, model selection using F1-score and fitting a simple logistic regression classifier

[7]:
```python
# Define the range of components to test (broad range from 50 to 1500 with steps␣
↪of 100)
component_range = list(range(500, 1501, 100))  # Test from 500 to 1500 with␣
↪steps of 100

# Initialize lists to store results
f1_scores = []

# Define the scorer for macro F1 score
f1_scorer = make_scorer(f1_score, average='macro')

# Loop over different values for n_components with tqdm to show progress
for n_components in tqdm(component_range, desc="Testing SVD Components"):
    # Apply Truncated SVD with n_components
    svd = TruncatedSVD(n_components=n_components, random_state=42)
    X_reduced = svd.fit_transform(data_train_tfidf)

    # Initialize Logistic Regression
    clf = LogisticRegression(max_iter=1000, random_state=42)

    # Perform 5-fold cross-validation and calculate the average macro F1 score
    scores = cross_val_score(clf, X_reduced, labels_train, cv=5,␣
↪scoring=f1_scorer)
    f1_scores.append(scores.mean())

    # Print the result for this number of components
    print(f"n_components={n_components}, Macro F1 Score={scores.mean()}")

# After loop, find the best number of components
best_n_components = component_range[np.argmax(f1_scores)]
print(f"Best number of components: {best_n_components}")
print(f"Best Macro F1 score: {max(f1_scores)}")
```

```
Testing SVD Components:   9%|
```

```
| 1/11 [00:10<01:41, 10.19s/it]

n_components=500, Macro F1 Score=0.6504907660796119

Testing SVD Components:  18%|
| 2/11 [00:34<02:45, 18.40s/it]

n_components=600, Macro F1 Score=0.6535555040606469

Testing SVD Components:  27%|
| 3/11 [00:58<02:48, 21.11s/it]

n_components=700, Macro F1 Score=0.6517923665849781

Testing SVD Components:  36%|
| 4/11 [01:29<02:55, 25.05s/it]

n_components=800, Macro F1 Score=0.6513715879166007

Testing SVD Components:  45%|
| 5/11 [02:02<02:47, 27.85s/it]

n_components=900, Macro F1 Score=0.6495969182621277

Testing SVD Components:  55%|
| 6/11 [02:39<02:35, 31.01s/it]

n_components=1000, Macro F1 Score=0.6486245868592354

Testing SVD Components:  64%|
| 7/11 [03:40<02:42, 40.64s/it]

n_components=1100, Macro F1 Score=0.6488489480920974

Testing SVD Components:  73%|
| 8/11 [04:31<02:12, 44.03s/it]

n_components=1200, Macro F1 Score=0.6516869522987132

Testing SVD Components:  82%|
| 9/11 [05:55<01:52, 56.40s/it]

n_components=1300, Macro F1 Score=0.6503540873568958

Testing SVD Components:
91%|     | 10/11
[06:52<00:56, 56.74s/it]

n_components=1400, Macro F1 Score=0.6488226545575827

Testing SVD Components:
100%|| 11/11
[07:55<00:00, 43.22s/it]

n_components=1500, Macro F1 Score=0.6508758865792623
Best number of components: 600
Best Macro F1 score: 0.6535555040606469
```

1.2.2 2) Keeping the 3rd best model (avoid over-fitting)

[9]:
```
# Apply Truncated SVD with 600 components
svd_best = TruncatedSVD(n_components=1200, random_state=42)
X_reduced_best = svd_best.fit_transform(data_train_tfidf)
```

1.2.3 3) Fitting different learning algorithms

[12]:
```
# Define the SVM model
svm_clf = LinearSVC(max_iter=1000, random_state=42)

# Perform 5-fold cross-validation for macro F1 score
svm_f1_scores = cross_val_score(svm_clf, X_reduced_best, labels_train, cv=5,␣
↪scoring=f1_scorer)

# Print the average macro F1 score across the 5 folds
print(f"Cross-validated Macro F1 Score (SVM): {svm_f1_scores.mean()}")
```

```
Cross-validated Macro F1 Score (SVM): 0.694617681179871
```

[17]:
```
# Initialize Multinomial Naive Bayes
nb_clf = MultinomialNB()

# Perform 5-fold cross-validation for macro F1 score
nb_f1_scores = cross_val_score(nb_clf, data_train, labels_train, cv=5,␣
↪scoring=make_scorer(f1_score, average='macro'))

# Print the average macro F1 score across the 5 folds
print(f"Cross-validated Macro F1 Score (Naive Bayes on counts): {nb_f1_scores.
↪mean()}")
```

```
Cross-validated Macro F1 Score (Naive Bayes on counts): 0.7104751581201816
```

[18]:
```
# Initialize Logistic Regression
log_clf = LogisticRegression(max_iter=1000, random_state=42)

# Perform 5-fold cross-validation for macro F1 score
log_f1_scores = cross_val_score(log_clf, X_reduced_best, labels_train, cv=5,␣
↪scoring=make_scorer(f1_score, average='macro'))

# Print the average macro F1 score across the 5 folds
print(f"Cross-validated Macro F1 Score (Logistic Regression): {log_f1_scores.
↪mean()}")
```

```
Cross-validated Macro F1 Score (Logistic Regression): 0.6516869522987132
```

[19]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize Random Forest
```

```
rf_clf = RandomForestClassifier(n_estimators=100, max_depth=15, random_state=42)

# Perform 5-fold cross-validation for macro F1 score
rf_f1_scores = cross_val_score(rf_clf, X_reduced_best, labels_train, cv=5,␣
↪scoring=make_scorer(f1_score, average='macro'))

# Print the average macro F1 score across the 5 folds
print(f"Cross-validated Macro F1 Score (Random Forest): {rf_f1_scores.mean()}")
```

```
Cross-validated Macro F1 Score (Random Forest): 0.43412083789333844
```

[20]:
```
import xgboost as xgb

# Initialize XGBoost
xgb_clf = xgb.XGBClassifier(n_estimators=100, max_depth=6, random_state=42)

# Perform 5-fold cross-validation for macro F1 score
xgb_f1_scores = cross_val_score(xgb_clf, X_reduced_best, labels_train, cv=5,␣
↪scoring=make_scorer(f1_score, average='macro'))

# Print the average macro F1 score across the 5 folds
print(f"Cross-validated Macro F1 Score (XGBoost): {xgb_f1_scores.mean()}")
```

```
Cross-validated Macro F1 Score (XGBoost): 0.6363559427423845
```

[21]:
```
from sklearn.neural_network import MLPClassifier

# Initialize MLP Classifier
mlp_clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42)

# Perform 5-fold cross-validation for macro F1 score
mlp_f1_scores = cross_val_score(mlp_clf, X_reduced_best, labels_train, cv=5,␣
↪scoring=make_scorer(f1_score, average='macro'))

# Print the average macro F1 score across the 5 folds
print(f"Cross-validated Macro F1 Score (MLP Classifier): {mlp_f1_scores.mean()}")
```

```
Cross-validated Macro F1 Score (MLP Classifier): 0.6850856935489185
```

[22]:
```
from sklearn.neighbors import KNeighborsClassifier

# Initialize KNN
knn_clf = KNeighborsClassifier(n_neighbors=5)

# Perform 5-fold cross-validation for macro F1 score
knn_f1_scores = cross_val_score(knn_clf, X_reduced_best, labels_train, cv=5,␣
↪scoring=make_scorer(f1_score, average='macro'))
```

```python
# Print the average macro F1 score across the 5 folds
print(f"Cross-validated Macro F1 Score (KNN): {knn_f1_scores.mean()}")
```

```
Cross-validated Macro F1 Score (KNN): 0.5468534858136735
```