Scientific journal recommender for submitting a publication

In [2]:
```
import string

folder = "datasets/"

from google.colab import drive
drive.mount('/content/drive')
folder = 'drive/MyDrive/Colab Notebooks/ScientificJournalRecommenderForSubmittingPublication/datasets/'
```

```
Mounted at /content/drive
```

Dataset

For each class (journal) there is a file in BibTeX format containing the articles published in that journal. Each file was cleaned and formatted with the following online tool BibTeX Tidy.

Each article is represented by a record with the following fields:

- **abstract**: Abstract of the article.
- **author**: Author of the article.
- **ENTRYTYPE**: Type of entry (article, book, inproceedings, etc.).
- **doi**: Digital Object Identifier of the article.
- **ID**: Unique identifier of the article.
- **issn**: International Standard Serial Number of the journal in which the article was published.
- **journal**: Journal in which the article was published.
- **keywords**: Keywords of the article.
- **note**: Additional information about the article.
- **pages**: Pages of the article.
- **title**: Title of the article.
- **url**: URL of the article.
- **volume**: Volume of the journal in which the article was published.
- **year**: Year of publication of the article.

The goal is to create a model that is able to predict the **journal** in which it will be published.

In [4]:
```
import os
import bibtexparser
import pandas as pd

# 1, 3, 4, 5
def read_bib_to_dataframe(file_path):
    #with open(file_path, 'r', encoding='utf-8') as bibtex_file:
    with open(file_path, 'r', encoding='latin-1') as bibtex_file:
        return bibtexparser.load(bibtex_file)

for filename in os.listdir(folder):
    if filename.endswith(".bib"):
        filename_path = os.path.join(folder, filename)
        bib_data = read_bib_to_dataframe(filename_path)
        if bib_data.entries:
            df = pd.DataFrame(bib_data.entries)
            df.to_csv(os.path.splitext(filename_path)[0] + '.csv', index=False)
        else:
            print("Error: ", filename, " is empty")
```

In [5]:
```
import pandas as pd
import os

dfs = []
for filename in os.listdir(folder):
    if filename.endswith(".csv"):
        dfs.append(pd.read_csv(os.path.join(folder, filename)))

df = pd.concat(dfs, ignore_index=True)

# Use the following id features to remove duplicates
for feature in ['doi', 'ID']:
    # Remove duplicates based on the subset of non-null, non-na, and non-empty values in the feature
```

```
    tmp_df = df[df[feature].notnull() & df[feature].notna() & (df[feature] != '')]
    duplicates_count = tmp_df.duplicated(subset=[feature]).sum()

    if duplicates_count > 0:
        print(f'Duplicates found using {feature}: {duplicates_count}\n\n')

        df = df[~df[feature].duplicated(keep='first') | df[feature].isnull() | df[feature].isna() | (df[feature]

print(df.info())
df.head()
df.to_csv(folder + 'all.csv', index=False)

for tmp_df in dfs:
    tmp_df = None
dfs = None
```

```
Duplicates found using ID: 63


<class 'pandas.core.frame.DataFrame'>
Int64Index: 10165 entries, 0 to 10227
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   abstract   10066 non-null  object
 1   keywords   10023 non-null  object
 2   author     10088 non-null  object
 3   url        10165 non-null  object
 4   doi        10165 non-null  object
 5   issn       10165 non-null  object
 6   year       10165 non-null  int64
 7   pages      10165 non-null  object
 8   volume     10165 non-null  int64
 9   journal    10165 non-null  object
 10  title      10165 non-null  object
 11  ENTRYTYPE  10165 non-null  object
 12  ID         10165 non-null  object
 13  note       39 non-null     object
dtypes: int64(2), object(12)
memory usage: 1.2+ MB
None
```

Feature Selection

The following features are selected:

- **abstract**: Abstract of the article.
- **keywords**: Keywords of the article.
- **title**: Title of the article.

The target feature is:

- **journal**: Journal in which the article was published.

In [3]:

```
# Removing unnecessary columns
import pandas as pd
df = pd.read_csv(folder + 'all.csv')

feature_names = ['abstract', 'keywords', 'title']
target_name = 'journal'

# Remove all the row that countains null values in the target_name column
df = df.dropna(subset=[target_name])

df = df[feature_names + [target_name]]
print(df.info())
df.head()

df.to_csv(folder + 'selected.csv', index=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10165 entries, 0 to 10164
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   abstract  10066 non-null  object
 1   keywords  10023 non-null  object
 2   title     10165 non-null  object
 3   journal   10165 non-null  object
dtypes: object(4)
memory usage: 317.8+ KB
None
```

In [7]:

```
# Cleaning data
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('omw-1.4')
nltk.download('wordnet')

language = 'english'
# Convert to lowercase
df[feature_names] = df[feature_names].applymap(lambda x: str(x).lower())
# Remove stopwords
nltk.download('stopwords')
stopwords_list = stopwords.words(language)
df[feature_names] = df[feature_names].apply(lambda x: x.apply(lambda words: ' '.join([w for w in words.split() if
# Remove punctuation
nltk.download('punkt')
df[feature_names] = df[feature_names].apply(lambda x: x.str.translate(str.maketrans('', '', string.punctuation)))
# Stemming
stemmer = nltk.stem.SnowballStemmer(language=language)
df[feature_names] = df[feature_names].apply(lambda x: x.apply(lambda words: ' '.join([stemmer.stem(w) for w in wo
# Tokenize
df[feature_names] = df[feature_names].apply(lambda x: x.apply(nltk.word_tokenize))

df.head()
df.to_csv(folder + 'selected_cleaned.csv', index=False)
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Text Representation

Put all the words of the selected features in a single column.

In [9]:

```
import pandas as pd

#folder = "datasets/"
#from google.colab import drive
#drive.mount('/content/drive')
#folder = 'drive/MyDrive/Colab Notebooks/ScientificJournalRecommenderForSubmittingPublication/datasets/'

df = pd.read_csv(folder + 'selected_cleaned.csv')


target_name = 'journal'

feature_names = df.columns.tolist()
feature_names.remove(target_name)

df['X'] = df[feature_names[0]]
for i in range(1, len(feature_names)):
    df['X'] = df['X'] + df[feature_names[i]]

# Remove null values
df = df.dropna(subset=['X'])

# Encode target_name
labels = df[target_name].unique()
df['y'] = df[target_name].replace(labels, list(range(len(labels))))

# Remove unnecessary columns
df = df[['X', 'y']]
print(df.info())
```

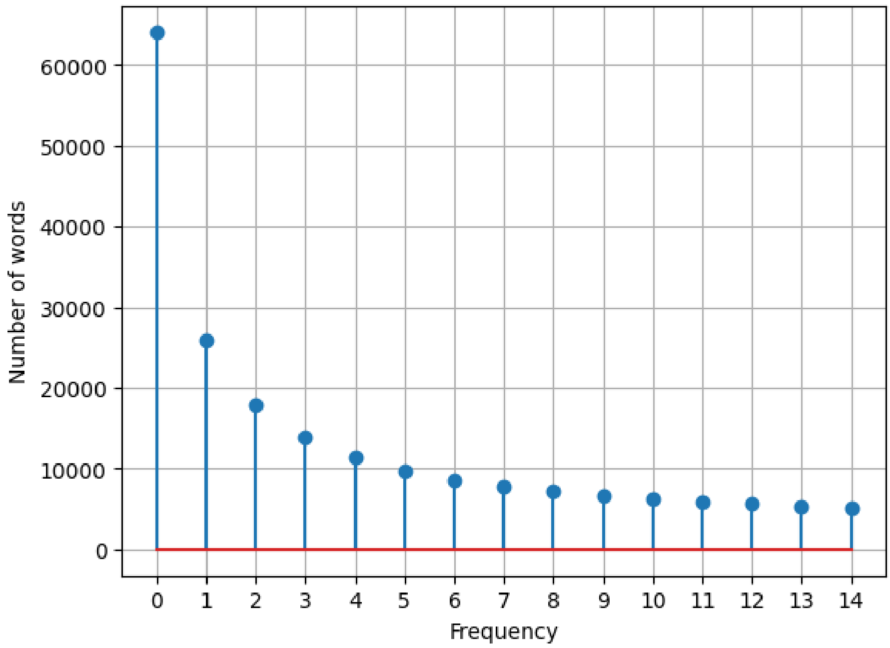
```
df.head()
df.to_csv(folder + 'selected_cleaned_combined_text.csv', index=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10165 entries, 0 to 10164
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   X       10165 non-null  object
 1   y       10165 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 159.0+ KB
None
```

In [59]:

```
from collections import Counter
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv(folder + 'selected_cleaned_combined_text.csv')

# Count frequency of each unique word in the dataset
word_counts = Counter()
for row in df['X']:
    word_counts.update(row.split())

def word_frequency(word_counts, max_frequency):
    frequency_data = [len(word_counts)]

    print("Number of unique words:", frequency_data[0])
    for frequency in range(1, max_frequency):
        number = sum(count > frequency for count in word_counts.values())
        print(f"Number of words with frequency = {frequency}: {number}")
        frequency_data.append(number)
    return frequency_data

max_frequency = 15


# Number words for each frequency
number_words_frequency = word_frequency(word_counts, max_frequency)

plt.stem(range(0, max_frequency), number_words_frequency)
plt.xlabel("Frequency")
plt.xticks(range(0, max_frequency))
plt.ylabel("Number of words")
plt.grid(True)
plt.show()
```

```
Number of unique words: 64055
Number of words with frequency = 1: 25899
Number of words with frequency = 2: 17832
Number of words with frequency = 3: 13812
Number of words with frequency = 4: 11331
Number of words with frequency = 5: 9713
Number of words with frequency = 6: 8634
Number of words with frequency = 7: 7788
Number of words with frequency = 8: 7154
Number of words with frequency = 9: 6691
Number of words with frequency = 10: 6285
Number of words with frequency = 11: 5933
Number of words with frequency = 12: 5667
Number of words with frequency = 13: 5413
Number of words with frequency = 14: 5204
```



Divide the dataset into training and test sets

In [60]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import gc

df_train, df_test = train_test_split(df, train_size=0.8, random_state=42)
# Delete df to free memory
df = None
gc.collect()
```

Out[60]:
```
63117
```

Bag of Words

The **Bag of Words** is a representation of a text that describes the occurrence of words within a document. The Bag of Words is created using the following methods:

- **CountVectorizer**: It counts the number of times a word appears in a document.
- **TfidfVectorizer**: It counts the number of times a word appears in a document, but it also takes into account the frequency of the word in the entire corpus.

Parameters:

- **max_features**: build a vocabulary that only consider the top max_features ordered by term frequency across the corpus.

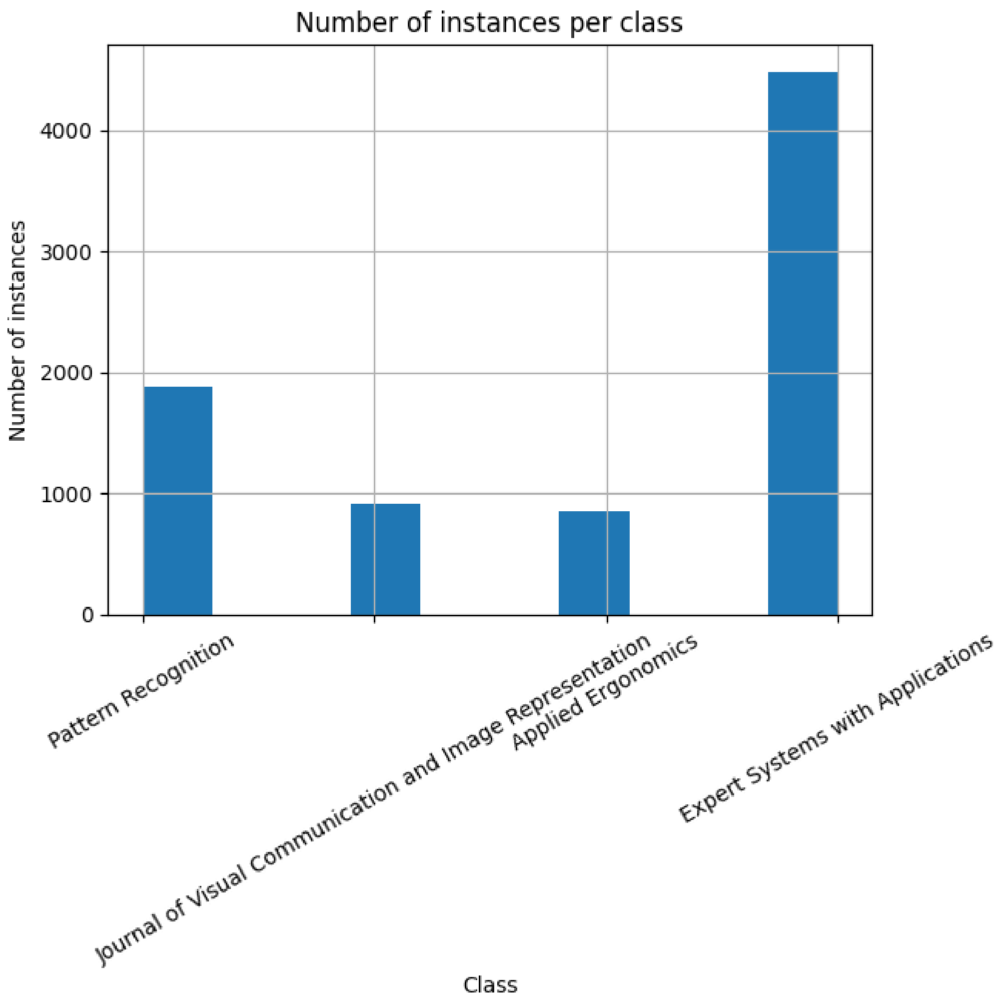
In [61]:
```
from sklearn.feature_extraction.text import CountVectorizer

bow = CountVectorizer(max_features=1000)

X_train = bow.fit_transform(df_train['X']).toarray()
y_train = df_train['y']

X_test = bow.transform(df_test['X']).toarray()
y_test = df_test['y']
```

Clasification

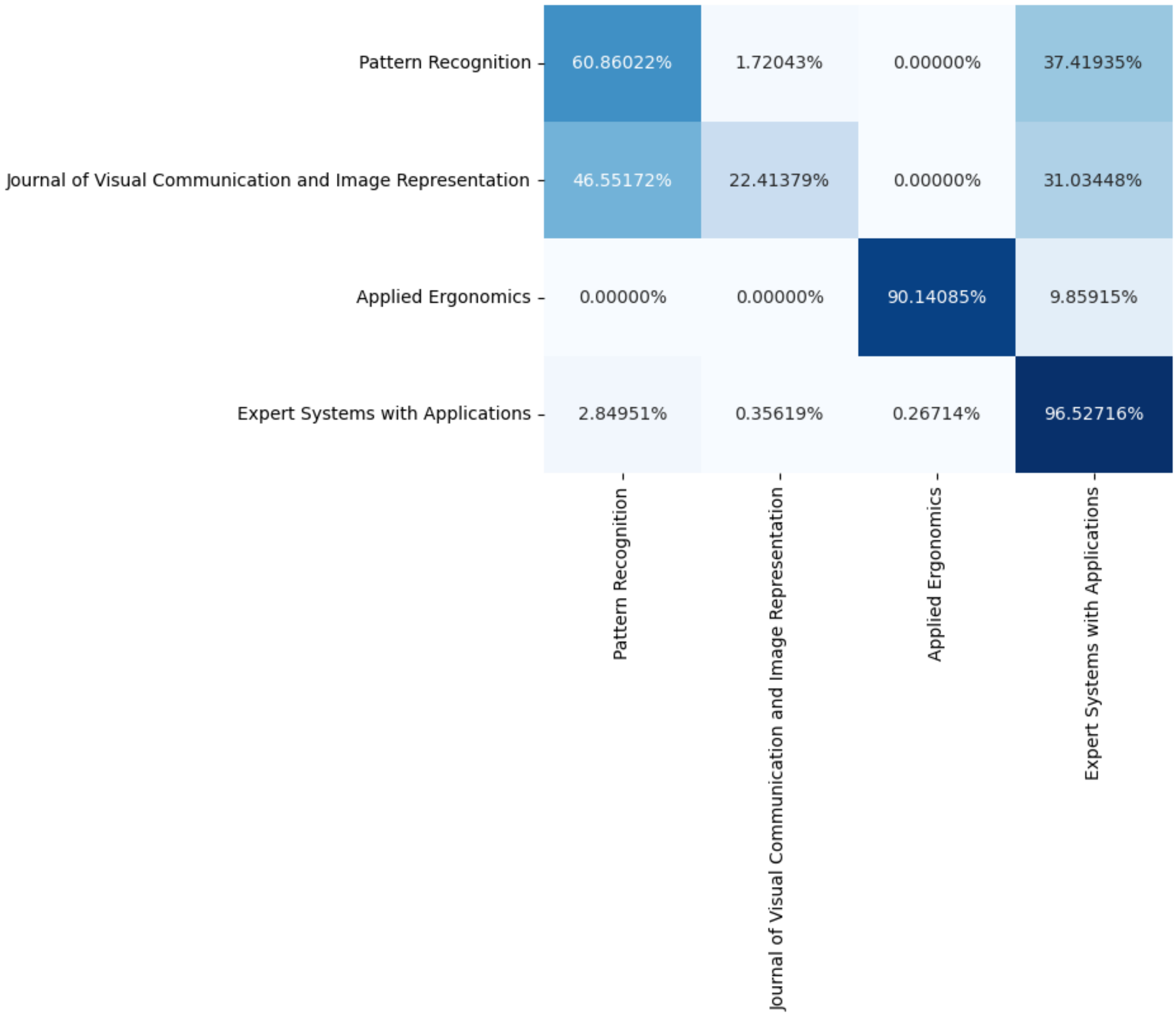
The classification is performed using the **Random Forest** algorithm. It is an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

In [62]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```python
from sklearn.ensemble import RandomForestClassifier

def plot_class_distribution(y):
    plt.hist(y)
    plt.title("Number of instances per class")
    plt.xlabel("Class")
    plt.ylabel("Number of instances")
    plt.xticks(list(range(len(labels))), labels)
    plt.xticks(rotation=30)
    plt.grid(True)
    plt.show()

def performance(y_test, y_pred):
    print(classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    cm = cm.astype('float')/cm.sum(axis=1)[:,np.newaxis]

    sns.heatmap(cm, annot=True, fmt='.5%', cmap='Blues', yticklabels=labels, xticklabels=labels, cbar=False)
    plt.show()

plot_class_distribution(y_train)

# BoW ConuntVectorizer
cls = RandomForestClassifier()
cls.fit(X_train, y_train)
y_pred = cls.predict(X_test)

performance(y_test, y_pred)
```

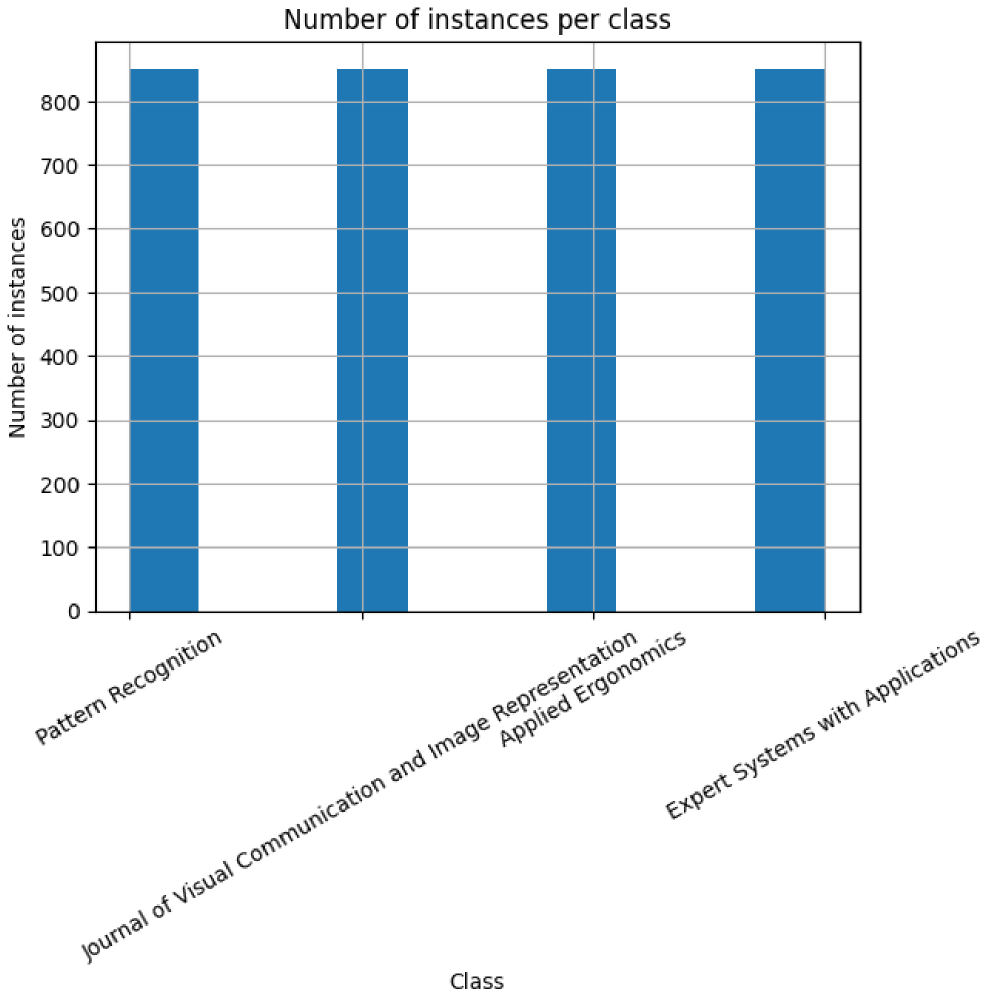



```
              precision    recall  f1-score   support

           0       0.67      0.61      0.64       465
           1       0.81      0.22      0.35       232
           2       0.98      0.90      0.94       213
           3       0.80      0.97      0.88      1123

    accuracy                           0.79      2033
   macro avg       0.82      0.67      0.70      2033
weighted avg       0.79      0.79      0.77      2033
```

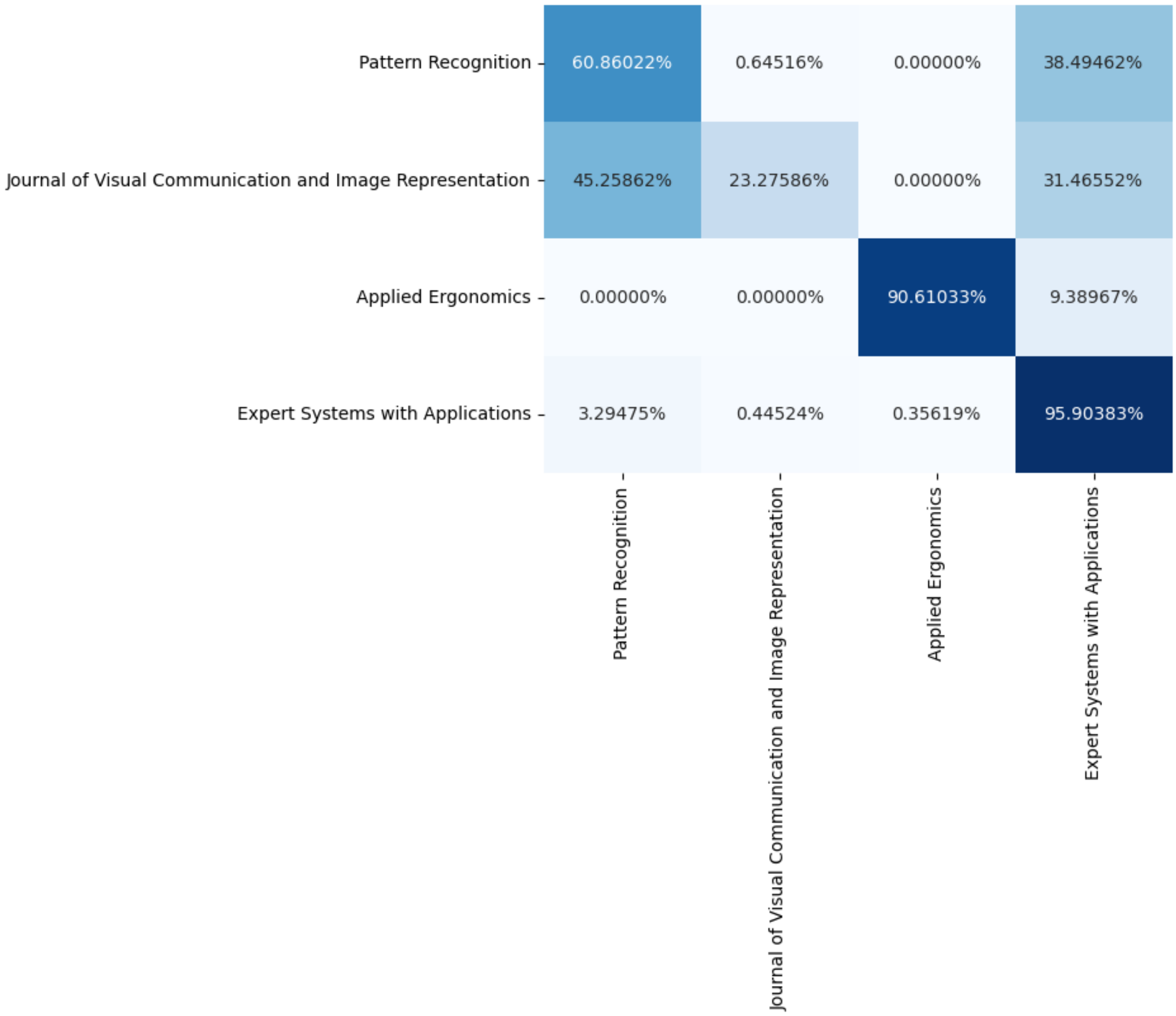


In [63]:

```
from imblearn.under_sampling import RandomUnderSampler
import gc

sampler = RandomUnderSampler(random_state=42)

# BoW ConuntVectorizer
X_train_under_sampled, y_train_under_sampled = sampler.fit_resample(X_train, y_train)
plot_class_distribution(y_train_under_sampled)

cls = RandomForestClassifier()
cls.fit(X_train, y_train)
y_pred = cls.predict(X_test)

performance(y_test, y_pred)

#Clean memory
X_train_under_sampled = None
y_train_under_sampled = None
gc.collect()
```

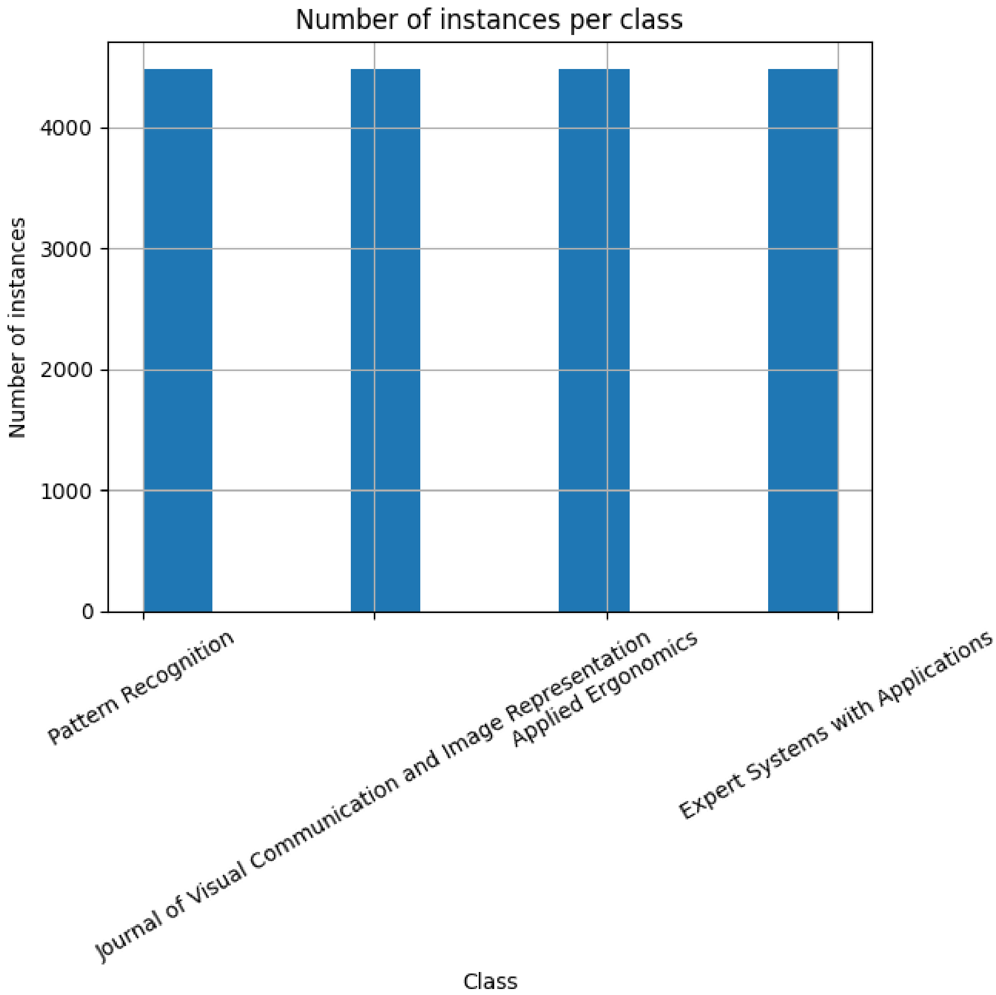


	precision	recall	f1-score	support
0	0.67	0.61	0.64	465
1	0.87	0.23	0.37	232
2	0.98	0.91	0.94	213
3	0.80	0.96	0.87	1123
accuracy			0.79	2033
macro avg	0.83	0.68	0.70	2033
weighted avg	0.80	0.79	0.77	2033

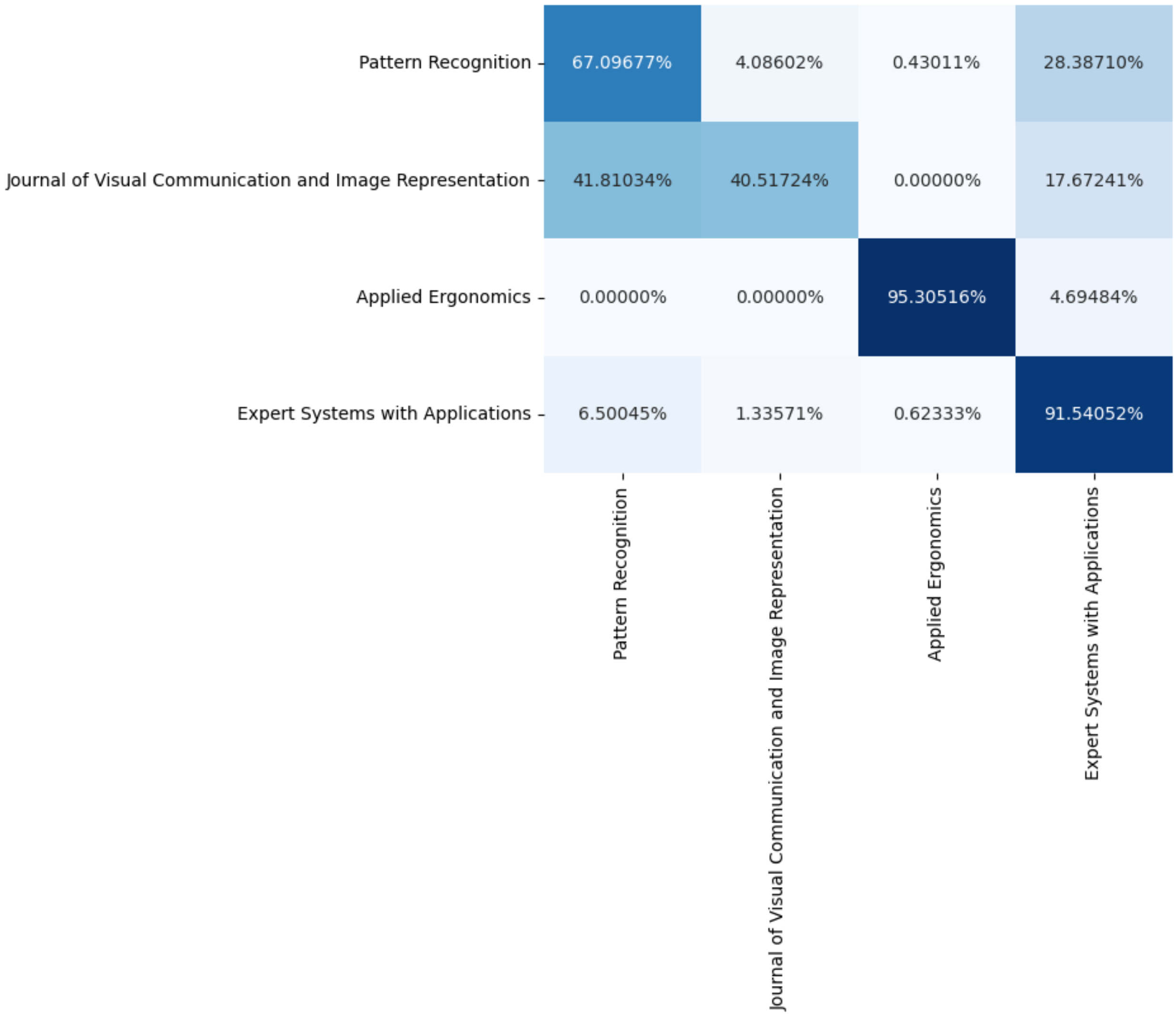


Out[63]: 12696

In [64]:
```
from imblearn.over_sampling import RandomOverSampler
import gc

sampler = RandomOverSampler(random_state=42)

X_train_over_sampled, y_train_over_sampled = sampler.fit_resample(X_train, y_train)
plot_class_distribution(y_train_over_sampled)

cls = RandomForestClassifier()
cls.fit(X_train_over_sampled, y_train_over_sampled)
y_pred = cls.predict(X_test)

performance(y_test, y_pred)

X_train_over_sampled = None
y_train_over_sampled = None
gc.collect()
```



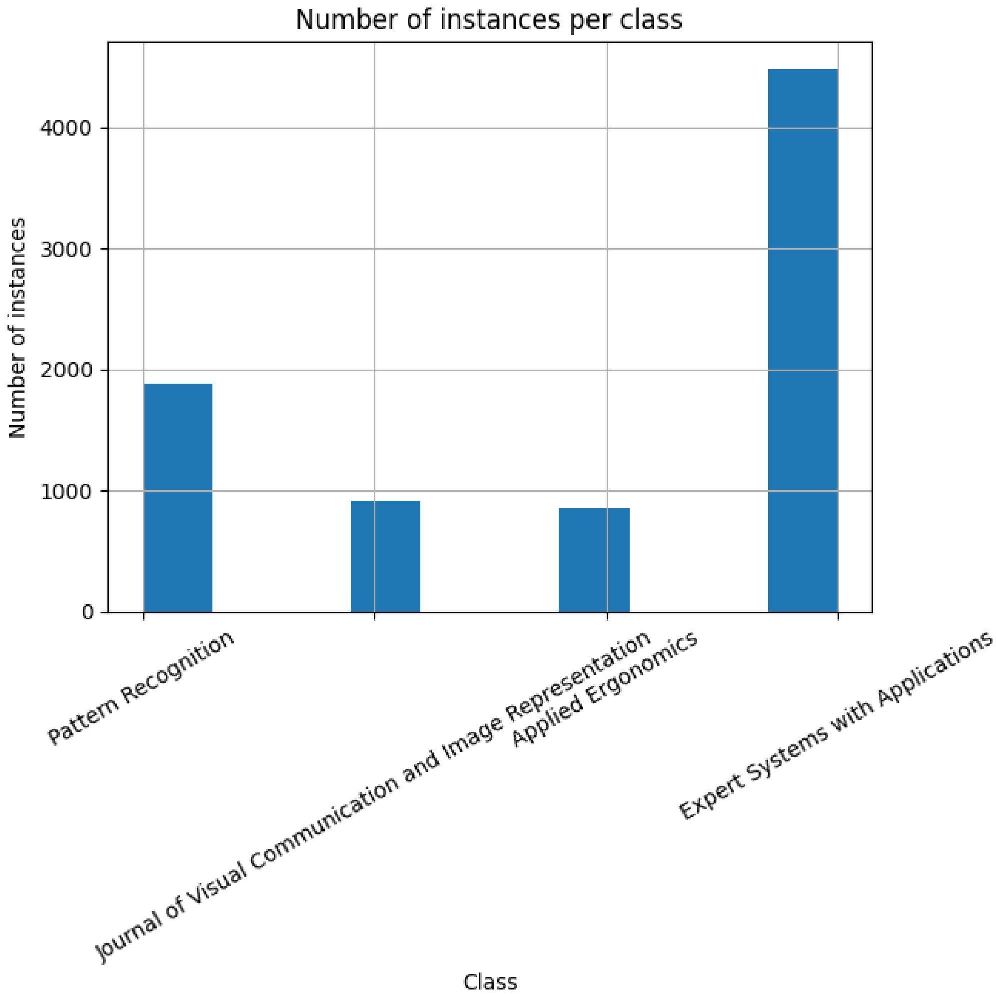
	precision	recall	f1-score	support
0	0.65	0.67	0.66	465
1	0.73	0.41	0.52	232
2	0.96	0.95	0.96	213
3	0.85	0.92	0.88	1123
accuracy			0.81	2033
macro avg	0.80	0.74	0.75	2033
weighted avg	0.80	0.81	0.80	2033



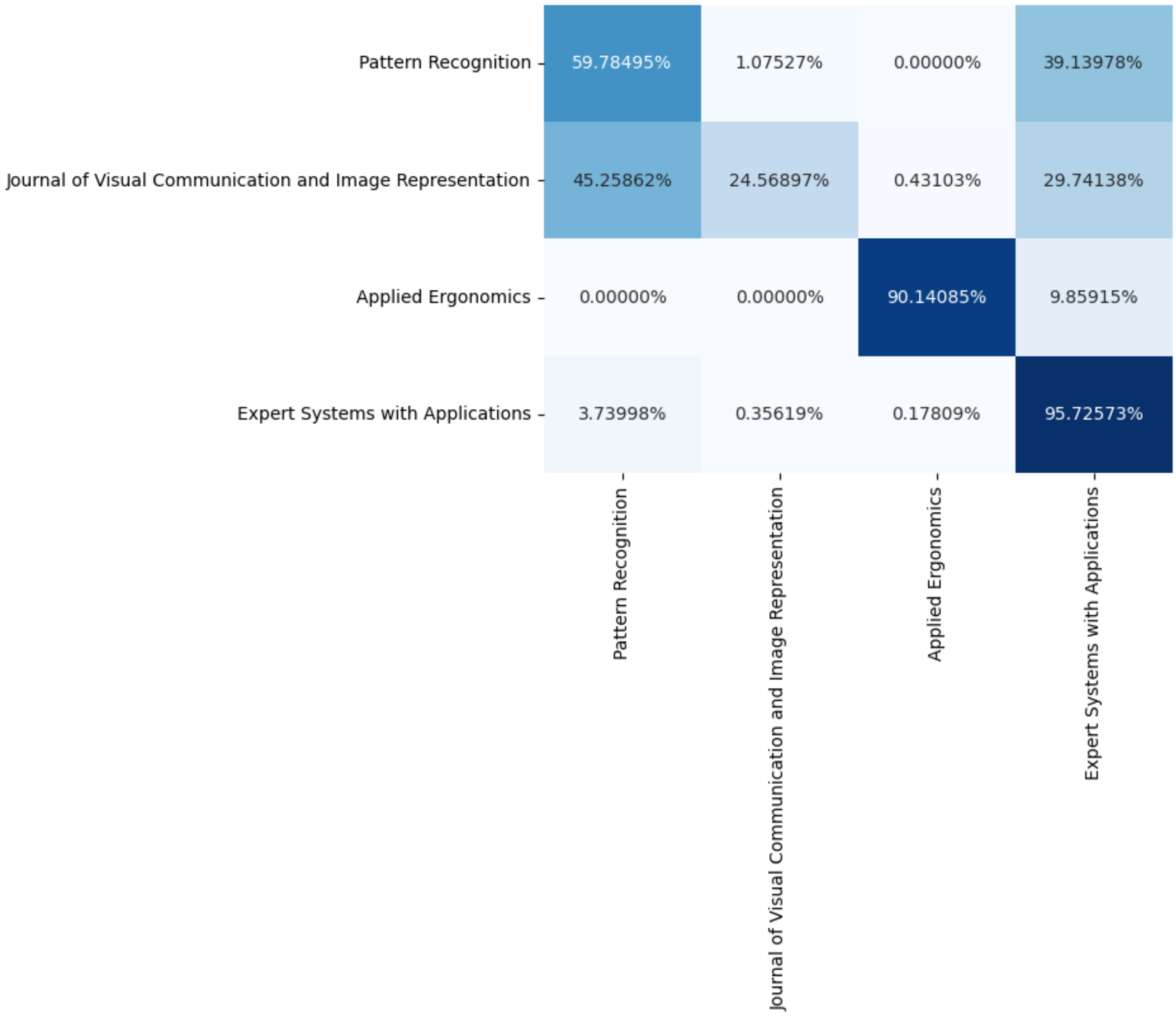
Out[64]: 3550

In [65]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

bow = TfidfVectorizer(max_features=1000)

X_train = bow.fit_transform(df_train['X']).toarray()
y_train = df_train['y']

X_test = bow.transform(df_test['X']).toarray()
y_test = df_test['y']
```

In [66]:
```
plot_class_distribution(y_train)

cls = RandomForestClassifier()
cls.fit(X_train, y_train)
y_pred = cls.predict(X_test)

performance(y_test, y_pred)
```

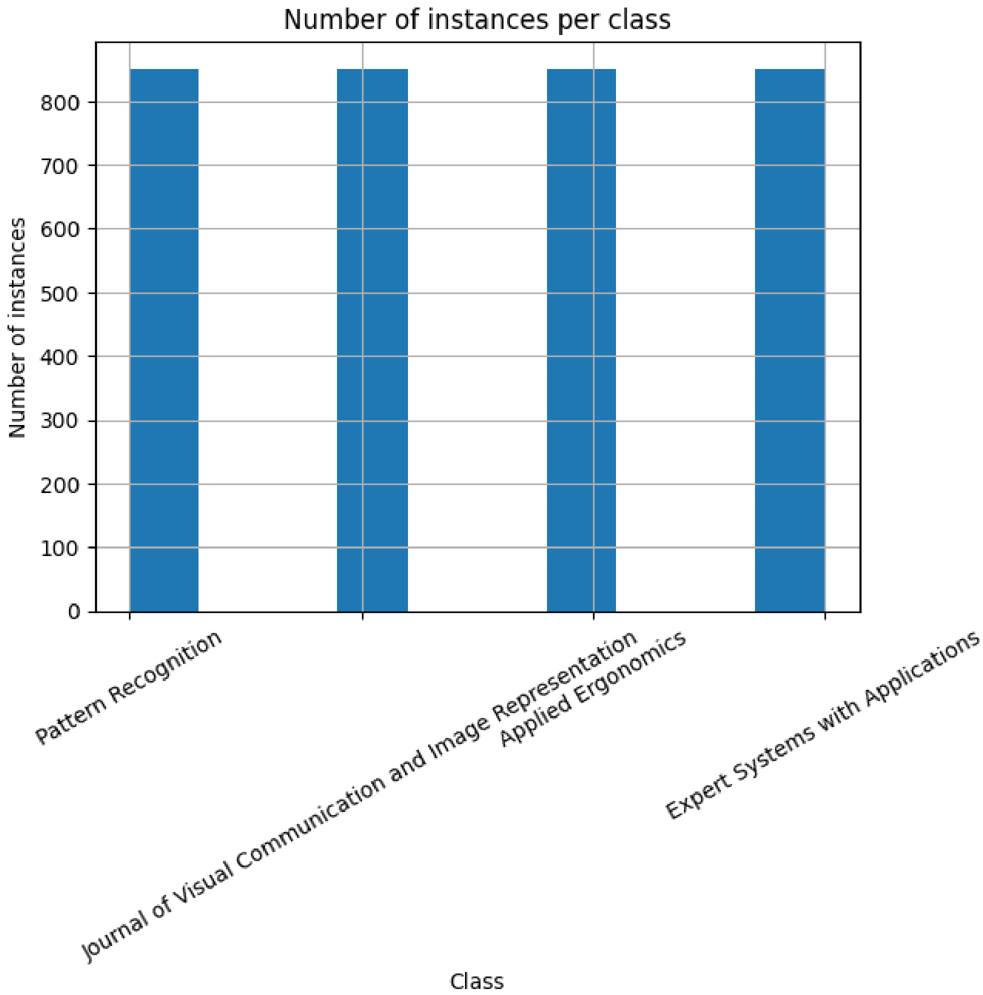


	precision	recall	f1-score	support
0	0.65	0.60	0.62	465
1	0.86	0.25	0.38	232
2	0.98	0.90	0.94	213
3	0.80	0.96	0.87	1123
accuracy			0.79	2033
macro avg	0.83	0.68	0.70	2033
weighted avg	0.79	0.79	0.77	2033

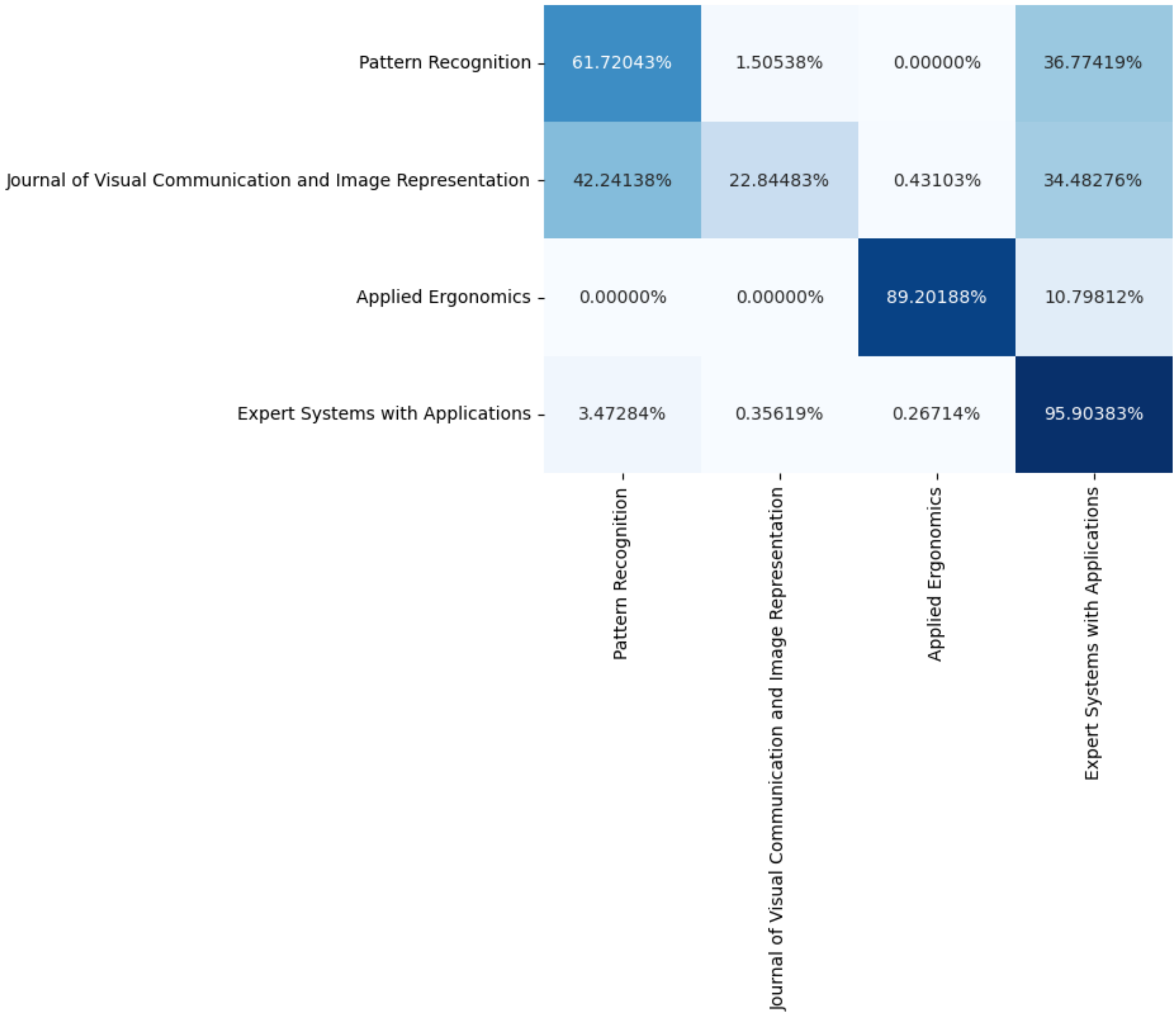


In [67]:

```
from imblearn.under_sampling import RandomUnderSampler
import gc

sampler = RandomUnderSampler(random_state=42)
X_train_under_sampled, y_train_under_sampled = sampler.fit_resample(X_train, y_train)
plot_class_distribution(y_train_under_sampled)

cls = RandomForestClassifier()
cls.fit(X_train, y_train)
y_pred = cls.predict(X_test)

performance(y_test, y_pred)

X_train_under_sampled = None
y_train_under_sampled = None
gc.collect()
```



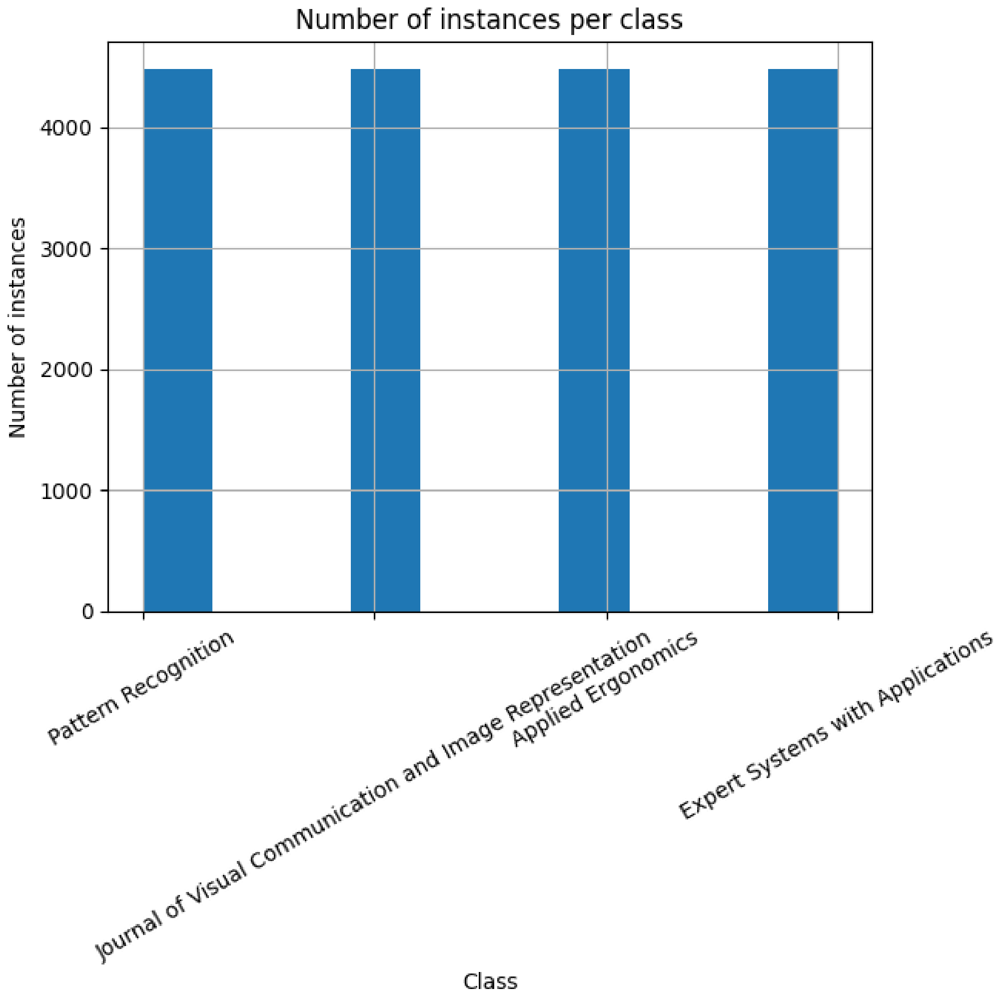
	precision	recall	f1-score	support
0	0.68	0.62	0.65	465
1	0.83	0.23	0.36	232
2	0.98	0.89	0.93	213
3	0.80	0.96	0.87	1123
accuracy			0.79	2033
macro avg	0.82	0.67	0.70	2033
weighted avg	0.79	0.79	0.77	2033



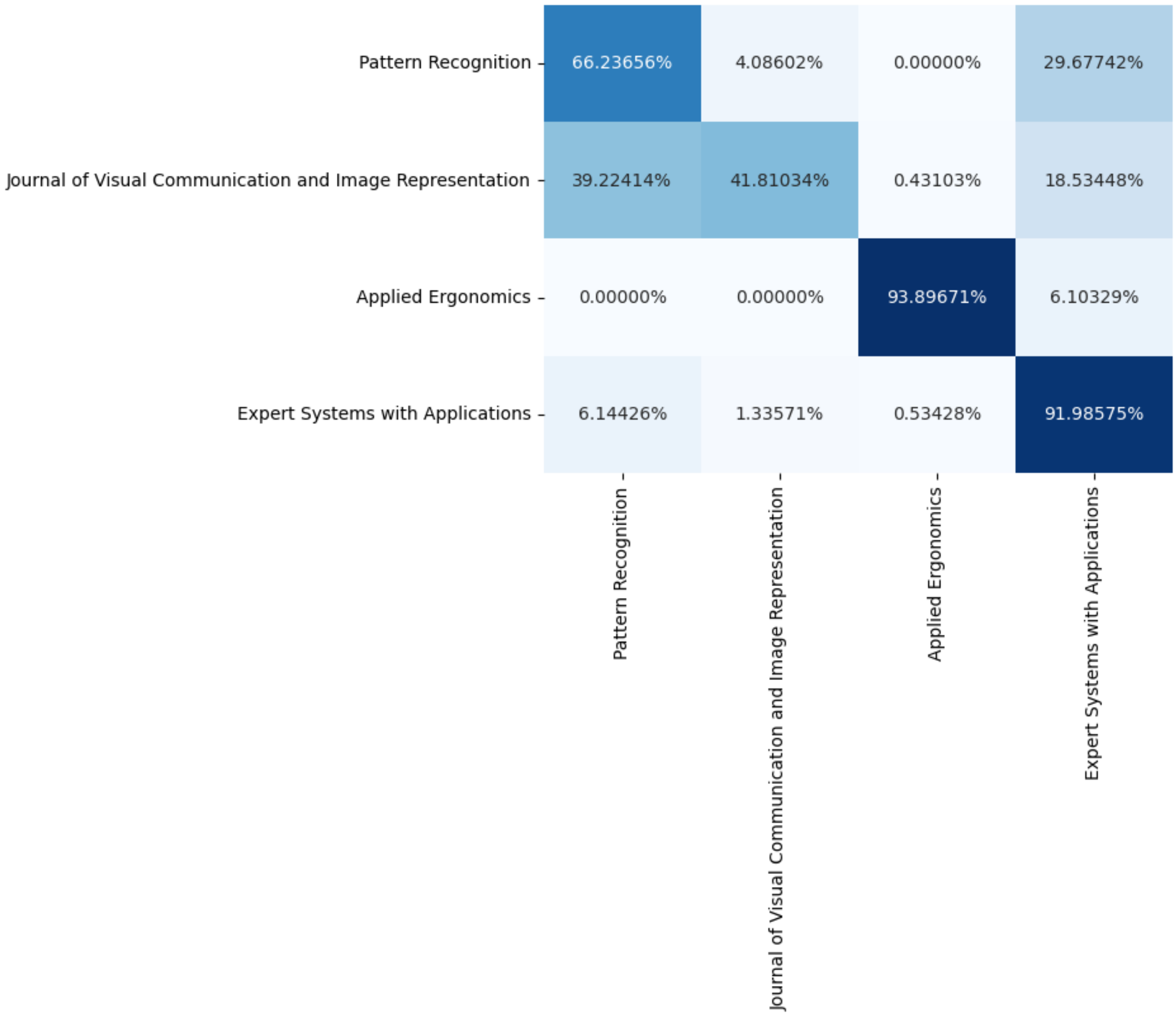
Out[67]: 10614

In [68]:

```
from imblearn.over_sampling import RandomOverSampler
import gc

sampler = RandomOverSampler(random_state=42)
X_train_over_sampled, y_train_over_sampled = sampler.fit_resample(X_train, y_train)
plot_class_distribution(y_train_over_sampled)

cls = RandomForestClassifier()
cls.fit(X_train_over_sampled, y_train_over_sampled)
y_pred = cls.predict(X_test)

performance(y_test, y_pred)

X_train = None
y_train = None
X_train_over_sampled = None
y_train_over_sampled = None
gc.collect()
```



	precision	recall	f1-score	support
0	0.66	0.66	0.66	465
1	0.74	0.42	0.53	232
2	0.97	0.94	0.95	213
3	0.84	0.92	0.88	1123
accuracy			0.81	2033
macro avg	0.80	0.73	0.76	2033
weighted avg	0.80	0.81	0.80	2033



Out[68]:

```
3524
```

Connectionist techniques

In this case, after pre-processing, a neural network based on an LSTM unit is trained.

In [86]:

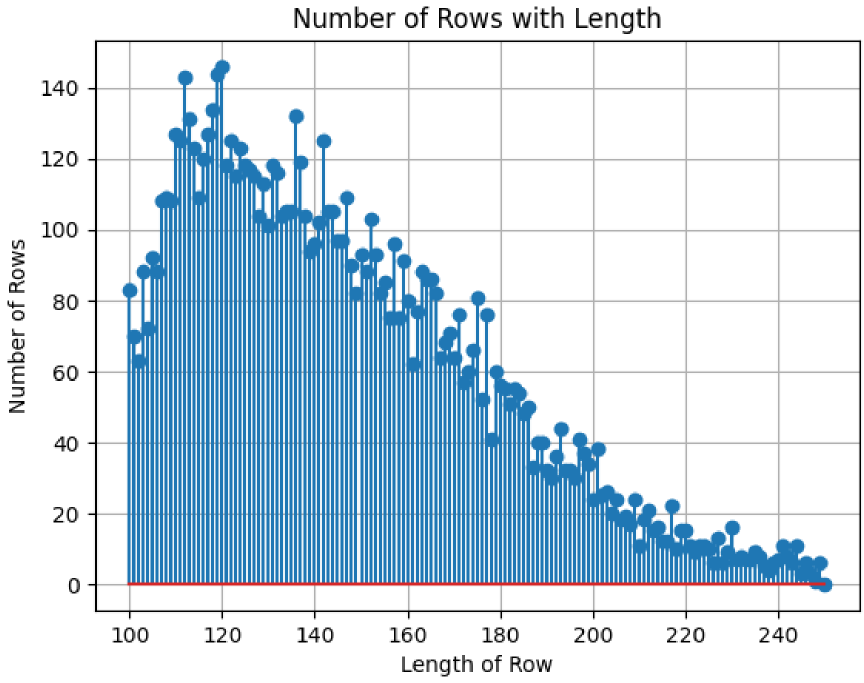
```
import tensorflow as tf
import pandas as pd
device = tf.config.list_physical_devices('GPU')

#from google.colab import drive
#drive.mount('/content/drive')
folder = 'drive/MyDrive/Colab Notebooks/ScientificJournalRecommenderForSubmittingPublication/datasets/'
#folder = 'datasets/'
df = pd.read_csv(folder + 'selected_cleaned_combined_text.csv')
df.info()
```

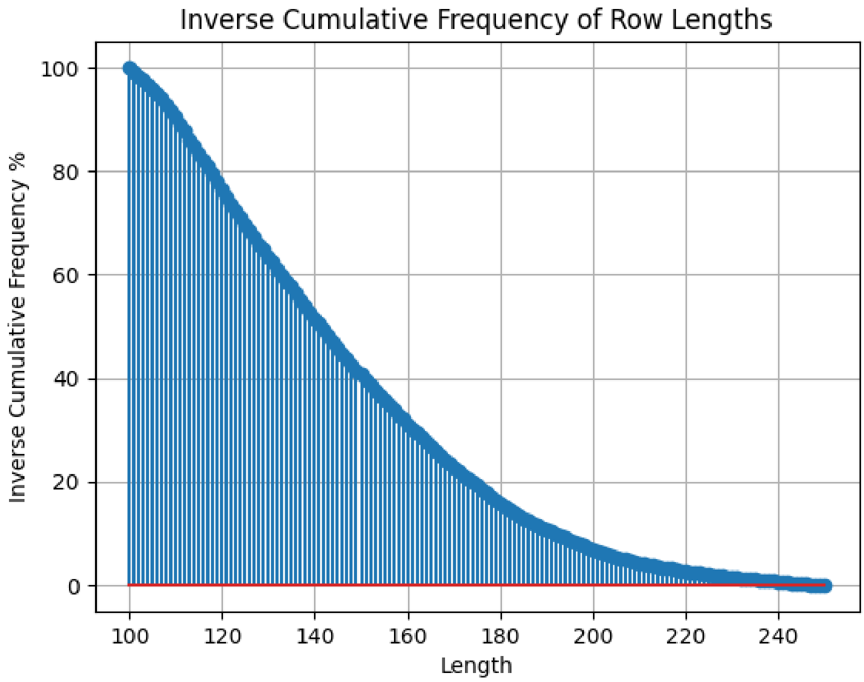
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10165 entries, 0 to 10164
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   X       10165 non-null  object
 1   y       10165 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 159.0+ KB
```

In [94]:

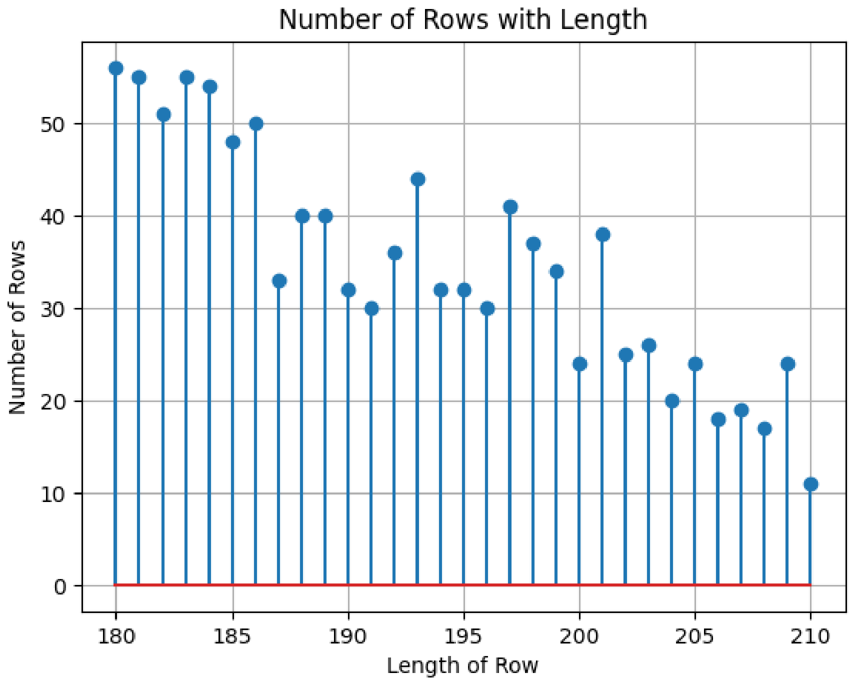
```
import matplotlib.pyplot as plt
from collections import Counter

lengths = [len(row.split()) for row in df['X']]
lengths_counts = Counter(lengths)

for i in range(0, max(lengths)):
    if i not in lengths_counts:
        lengths_counts[i] = 0
```

```python
lengths = sorted(lengths_counts.items())

def plot_lengths(lengths, length_range):
    # Filtering lengths
    lengths = [length for length in lengths if length_range[0] <= length[0] <= length_range[1]]

    plt.stem([length for length, count in lengths], [count for length, count in lengths])
    plt.title("Number of sequence with this Length")
    plt.xlabel("Length")
    plt.ylabel("Number of sequence")
    plt.grid(True)
    plt.show()

    inverse_cumulative = []
    inverse_cumulative_sum = sum(count for length, count in lengths)
    for length, count in lengths:
        inverse_cumulative.append(inverse_cumulative_sum)
        inverse_cumulative_sum -= count

    plt.stem([length for length, count in lengths], [count / inverse_cumulative[0]*100 for count in inverse_cumul
    plt.title("Inverse Cumulative Number of sequence with this Length")
    plt.xlabel("Length")
    plt.ylabel("Cumulative Number of sequence %")
    plt.grid(True)
    plt.show()


filter_lengths = [100, 250]
plot_lengths(lengths, filter_lengths)

filter_lengths = [180, 210]
plot_lengths(lengths, filter_lengths)
```

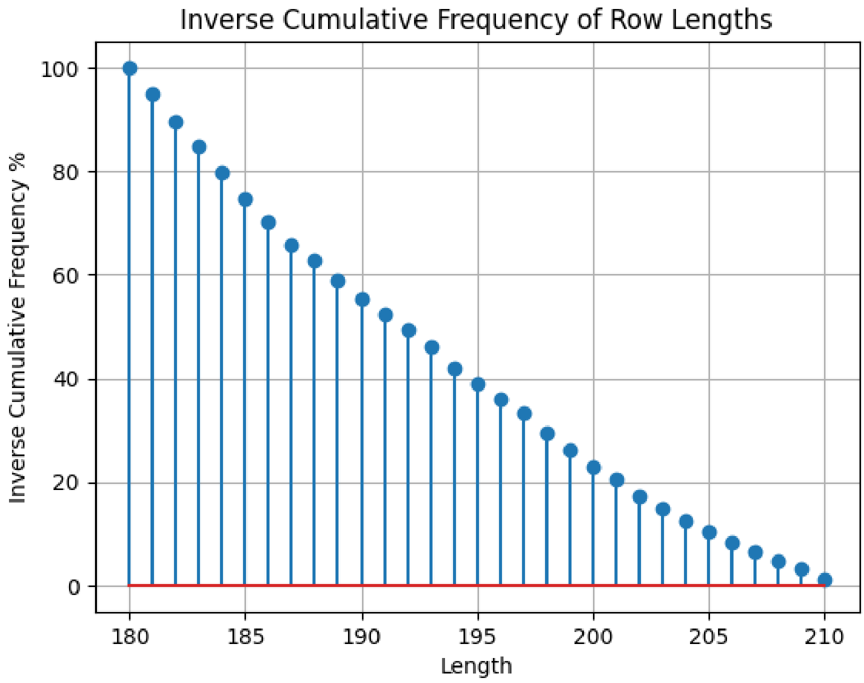




Inverse Cumulative Frequency of Row Lengths
Inverse Cumulative Frequency %
100
80
60
40
20
0
100
120
140
160
180
200
220
240
Length



Number of Rows with Length
Number of Rows
50
40
30
20
10
0
180
185
190
195
200
205
210
Length of Row



In [98]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing import sequence
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split

sequence_length = 200
X_train, X_test, y_train, y_test = train_test_split(df['X'], df['y'], test_size=0.2, random_state=42)

print(X_train.shape)
print(y_test.shape)

num_words = number_words_frequency[5]
print("Number words: ", num_words)
tokenizer = Tokenizer(num_words = number_words_frequency[5])
tokenizer.fit_on_texts(X_train)

print("Sequence length: ", sequence_length)
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_train_pad = sequence.pad_sequences(X_train_seq, maxlen=sequence_length)

X_test_seq = tokenizer.texts_to_sequences(X_test)
X_test_pad= sequence.pad_sequences(X_test_seq, maxlen=sequence_length)


num_classes = len(df['y'].unique())
# Convert your integer labels to one-hot encoded format
y_train_one_hot = to_categorical(y_train, num_classes=num_classes)
y_test_one_hot = to_categorical(y_test, num_classes=num_classes)
```

```
(8132,)
(2033,)
Number words:  9713
Sequence length:  200
```

In [102…
```
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
from keras import backend as K

def f1_score(y_true, y_pred):
    # Calculate precision and recall
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    actual_positives = K.sum(K.round(K.clip(y_true, 0, 1)))

    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (actual_positives + K.epsilon())

    # Calculate F1 score
    f1 = 2 * (precision * recall) / (precision + recall + K.epsilon())

    return f1
```

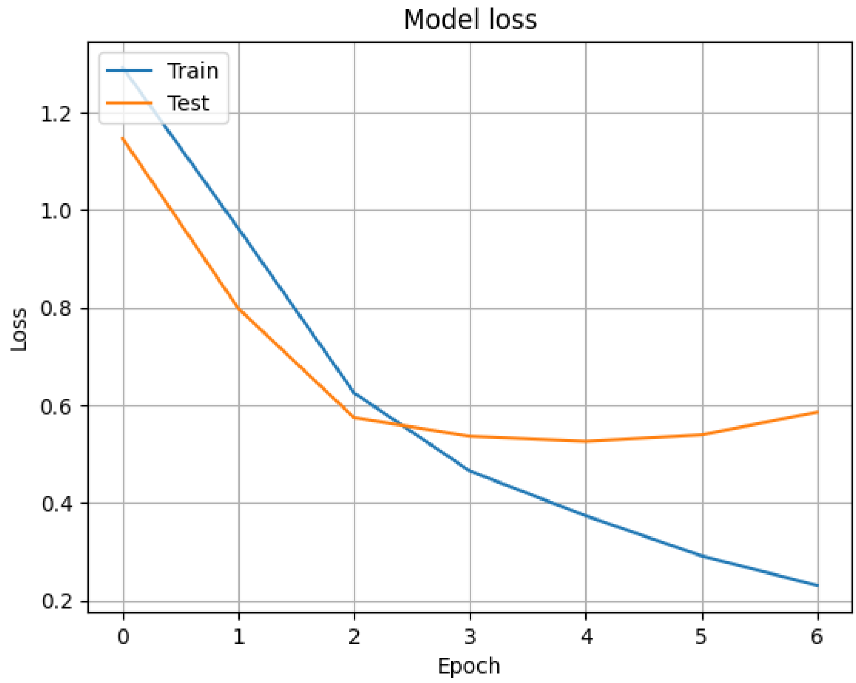
```python
def crete_model(num_words, embedding_dim, sequence_length, lstm_units, num_classes):
        model = Sequential()

        model.add(Embedding(input_dim=num_words, output_dim=embedding_dim, input_length=sequence_length))
        model.add(LSTM(units=lstm_units, dropout=0.2, recurrent_dropout=0.2))
        model.add(Dense(units=num_classes, activation='softmax'))
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[f1_score])
        return model


embedding_dim = 200
lstm_units = 200

model = crete_model(num_words, embedding_dim, sequence_length, lstm_units, num_classes)
model.summary()
```

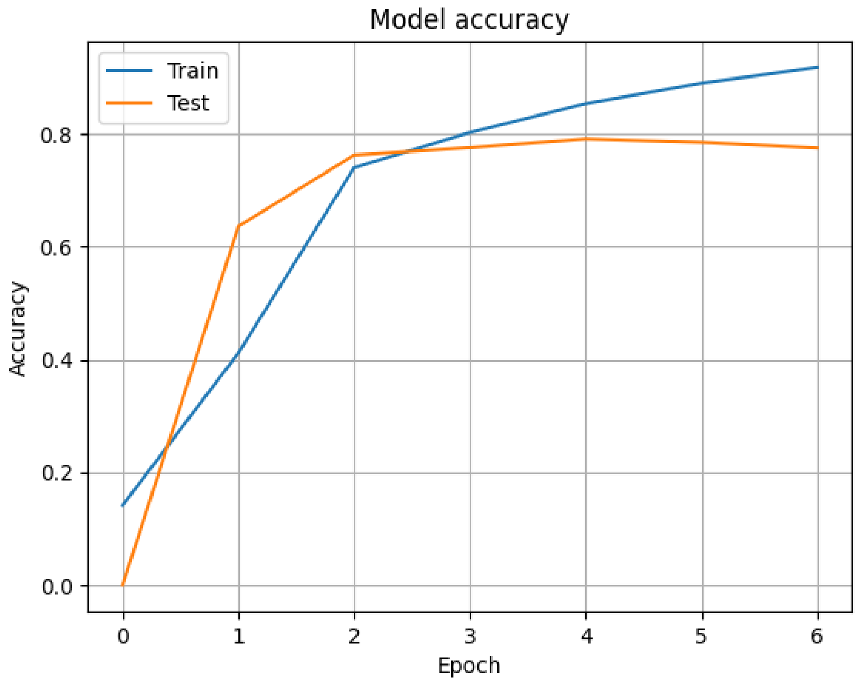
```
WARNING:tensorflow:Layer lstm_12 will not use cuDNN kernels since it doesn't meet the criteria. It will use a ge
neric GPU kernel as fallback when running on GPU.
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_12 (Embedding)    (None, 200, 200)          1942600

 lstm_12 (LSTM)              (None, 200)               320800

 dense_12 (Dense)            (None, 4)                 804

=================================================================
Total params: 2264204 (8.64 MB)
Trainable params: 2264204 (8.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [103…

```python
history = model.fit(X_train_pad, y_train_one_hot,
                    batch_size=512, epochs=10,
                    validation_data=(X_test_pad, y_test_one_hot),
                    callbacks=[
                        EarlyStopping(monitor='val_loss', patience=2)
                    ])

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.grid(True)
plt.show()

plt.plot(history.history['f1_score'])
plt.plot(history.history['val_f1_score'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.grid(True)
plt.show()
```

```
Epoch 1/10
16/16 [==============================] - 16s 768ms/step - loss: 1.2916 - f1_score: 0.1415 - val_loss: 1.1462 - v
al_f1_score: 0.0000e+00
Epoch 2/10
16/16 [==============================] - 11s 675ms/step - loss: 0.9619 - f1_score: 0.4118 - val_loss: 0.7979 - v
al_f1_score: 0.6366
Epoch 3/10
16/16 [==============================] - 12s 734ms/step - loss: 0.6251 - f1_score: 0.7407 - val_loss: 0.5745 - v
al_f1_score: 0.7631
Epoch 4/10
16/16 [==============================] - 12s 746ms/step - loss: 0.4652 - f1_score: 0.8031 - val_loss: 0.5365 - v
al_f1_score: 0.7763
Epoch 5/10
16/16 [==============================] - 14s 885ms/step - loss: 0.3736 - f1_score: 0.8542 - val_loss: 0.5258 - v
al_f1_score: 0.7916
Epoch 6/10
16/16 [==============================] - 12s 766ms/step - loss: 0.2916 - f1_score: 0.8906 - val_loss: 0.5394 - v
al_f1_score: 0.7856
Epoch 7/10
16/16 [==============================] - 12s 726ms/step - loss: 0.2310 - f1_score: 0.9186 - val_loss: 0.5859 - v
al_f1_score: 0.7757
```

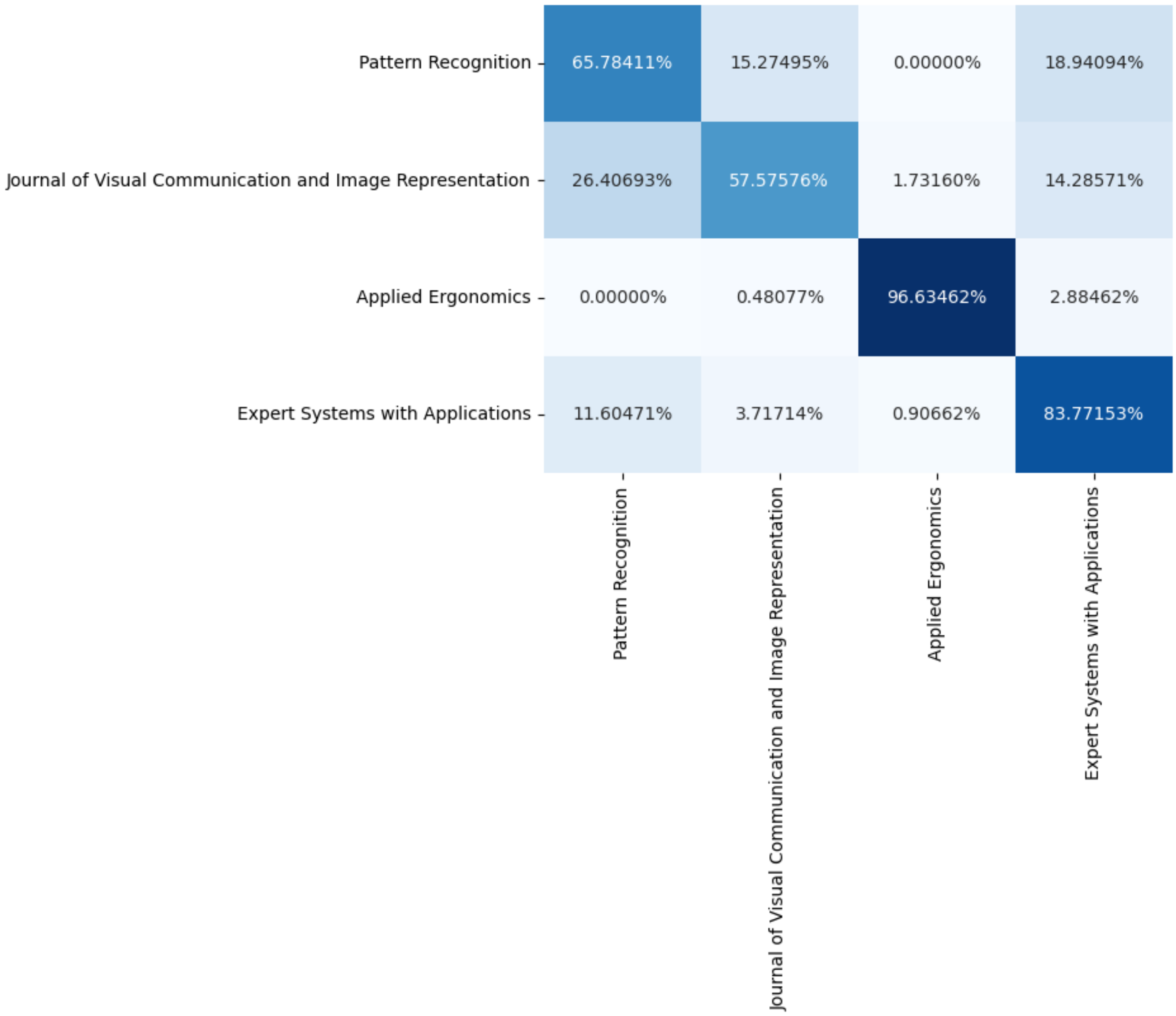




In [104…

```
# Evaluate the model on the test set using accuracy, f1-score, precision and recall
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

y_pred = model.predict(X_test_pad)
y_pred = np.argmax(y_pred, axis=1)

performance(y_test, y_pred)
```

```
64/64 [==============================] - 4s 55ms/step
              precision    recall  f1-score   support

           0       0.63      0.66      0.64       491
           1       0.53      0.58      0.55       231
           2       0.93      0.97      0.95       208
           3       0.88      0.84      0.86      1103

    accuracy                           0.78      2033
   macro avg       0.74      0.76      0.75      2033
weighted avg       0.78      0.78      0.78      2033
```


Pattern Recognition
Journal of Visual Communication and Image Representation
Applied Ergonomics
Expert Systems with Applications
65.78411%
15.27495%
0.00000%
18.94094%
26.40693%
57.57576%
1.73160%
14.28571%
0.00000%
0.48077%
96.63462%
2.88462%
11.60471%
3.71714%
0.90662%
83.77153%
Pattern Recognition
Journal of Visual Communication and Image Representation
Applied Ergonomics
Expert Systems with Applications