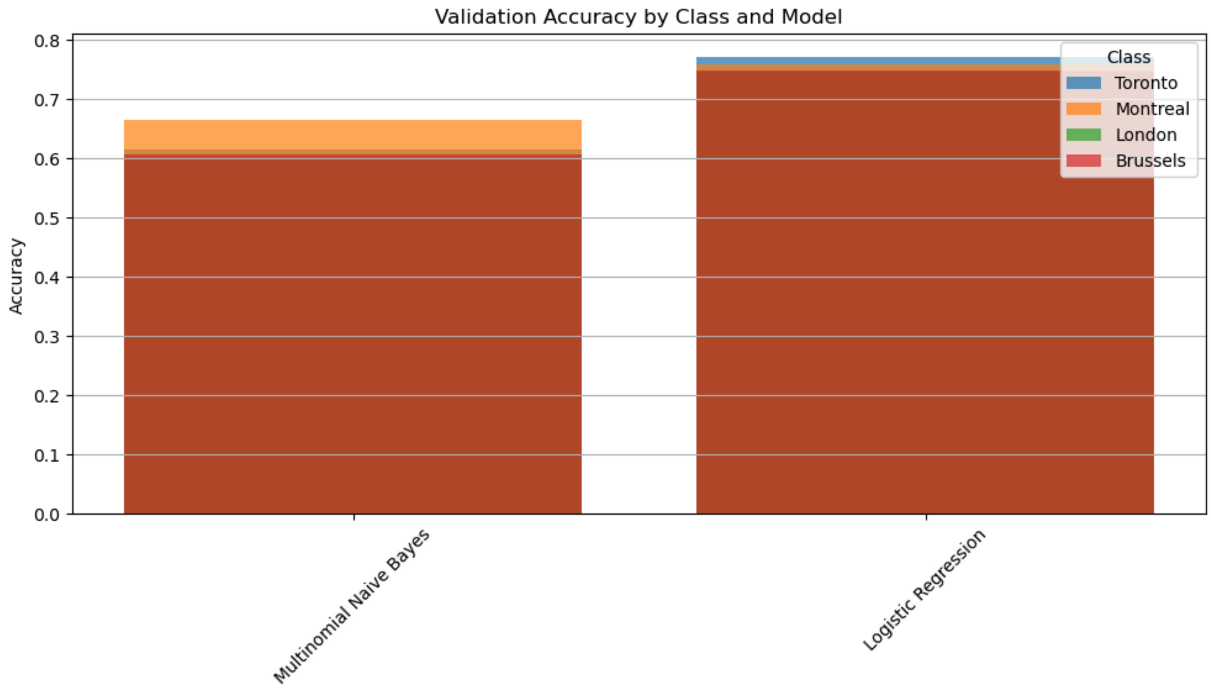
In [25]:
```python
import pandas as pd
import numpy as np
import time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score, GridSearchCV, StratifiedKFold,
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer, MinMaxScaler
from sentence_transformers import SentenceTransformer
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt

# Load training and test data
train_file = 'Train.csv'
test_file = 'Test.csv'
output_file = 'submissions.csv'

train_data = pd.read_csv(train_file, header=None, names=['text', 'subreddit'])
test_data = pd.read_csv(test_file)

# TF-IDF Vectorization with Stopwords and N-grams
tfidf_vectorizer = TfidfVectorizer(
    ngram_range=(1, 3),  # Use uni-, bi-, and tri-grams
    max_features=8000,  # Increase feature size
    stop_words='english'  # Remove common stopwords
)
tfidf_features = tfidf_vectorizer.fit_transform(train_data['text'])

# Dimensionality Reduction
svd = TruncatedSVD(n_components=400, random_state=42)
reduced_tfidf = svd.fit_transform(tfidf_features)

# Ensure non-negative features for MultinomialNB
minmax_scaler = MinMaxScaler()
non_negative_tfidf = minmax_scaler.fit_transform(reduced_tfidf)

# Normalize features for Logistic Regression
tfidf_normalizer = Normalizer(norm='l2')
normalized_train_tfidf = tfidf_normalizer.fit_transform(reduced_tfidf)

# Sentence Embeddings
sentence_model = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')
sentence_embeddings = sentence_model.encode(train_data['text'].tolist())

# Normalize Sentence Embeddings
sentence_normalizer = Normalizer(norm='l2')
normalized_train_sentences = sentence_normalizer.fit_transform(sentence_embeddings)

# Combine features
X_combined = np.hstack([
    normalized_train_tfidf,              # Normalized TF-IDF features
    normalized_train_sentences           # Normalized Sentence Embeddings
])
```

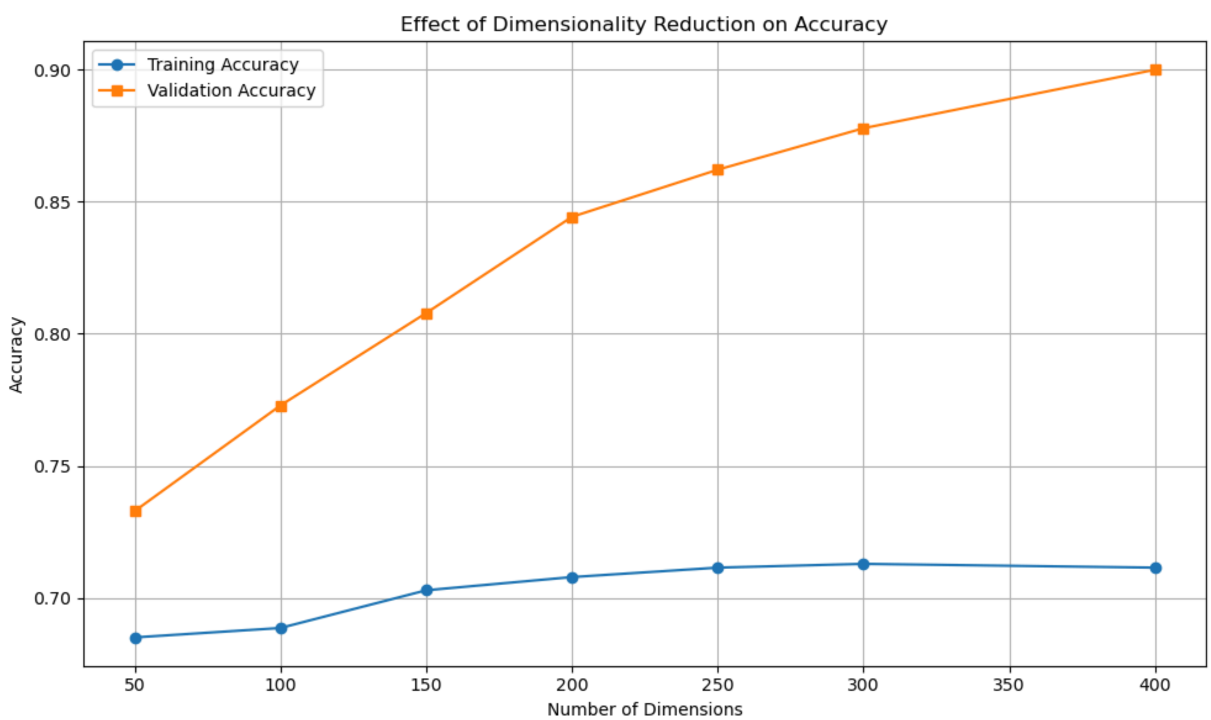
```python
# Map labels
label_map = {label: idx for idx, label in enumerate(train_data['subreddit'].unique(
y = train_data['subreddit'].map(label_map)

# Hyperparameter grids for fine-tuning
param_grid_nb = {
    'alpha': [0.01, 0.1, 0.5, 1.0]  # Explore more alpha values
}

param_grid_lr = {
    'C': [0.1, 1, 10, 50],  # Include higher C values to reduce regularization
    'solver': ['lbfgs']
}

kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Models and parameter search
models = {
    'Multinomial Naive Bayes': (MultinomialNB(), param_grid_nb, non_negative_tfidf)
    'Logistic Regression': (LogisticRegression(max_iter=3000), param_grid_lr, X_com
}

results = []
validation_accuracies = {
    'Model': [],
    'Class': [],
    'Accuracy': []
}

for model_name, (model, param_grid, X_features) in models.items():
    start_time = time.time()
    grid_search = GridSearchCV(
        model, param_grid=param_grid, cv=kf, scoring='accuracy', verbose=1, n_jobs=
    )
    grid_search.fit(X_features, y)

    # Best model and parameters
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_

    # Cross-validated predictions for classification report
    y_pred = cross_val_predict(best_model, X_features, y, cv=kf)
    class_report = classification_report(y, y_pred, target_names=list(label_map.key

    # Collect results
    results.append({
        'Model': model_name,
        'Training Accuracy': grid_search.best_score_,
        'Validation Accuracy (Class-wise)': {label: class_report[label]['f1-score']
        'Time (5-fold)': round(time.time() - start_time, 2),
        'Numb. Params': X_features.shape[1] if model_name == 'Multinomial Naive Bay
    })

    # Store validation accuracy for plotting
    for class_name, metrics in class_report.items():
```

```
        if class_name in label_map.keys():
            validation_accuracies['Model'].append(model_name)
            validation_accuracies['Class'].append(class_name)
            validation_accuracies['Accuracy'].append(metrics['f1-score'])

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

# Format Validation Accuracy for readability
results_df['Validation Accuracy (Class-wise)'] = results_df['Validation Accuracy (C
    lambda x: '\n'.join([f"{key}: {round(value, 4)}" for key, value in x.items()])
)
print(results_df)

# Plot grouped bar chart for validation accuracy
validation_df = pd.DataFrame(validation_accuracies)
plt.figure(figsize=(10, 6))
for class_name in validation_df['Class'].unique():
    class_data = validation_df[validation_df['Class'] == class_name]
    plt.bar(class_data['Model'], class_data['Accuracy'], label=class_name, alpha=0.

plt.title('Validation Accuracy by Class and Model')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
plt.legend(title='Class')
plt.grid(axis='y')
plt.tight_layout()
plt.savefig('validation_accuracy_by_class.png')
plt.show()

# Dimensionality reduction graph
dimensions = [50, 100, 150, 200, 250, 300, 400]
training_accuracies = []
validation_accuracies_dim = []

for dim in dimensions:
    svd = TruncatedSVD(n_components=dim, random_state=42)
    reduced_features = svd.fit_transform(tfidf_features)
    normalized_features = tfidf_normalizer.fit_transform(reduced_features)

    # Logistic Regression for dimension analysis
    model = LogisticRegression(max_iter=3000)
    scores = cross_val_score(model, normalized_features, y, cv=kf, scoring='accurac
    training_accuracies.append(scores.mean())

    # Train/test split for validation
    model.fit(normalized_features, y)
    validation_accuracies_dim.append(accuracy_score(y, model.predict(normalized_fea

# Plot graph
plt.figure(figsize=(10, 6))
plt.plot(dimensions, training_accuracies, label='Training Accuracy', marker='o')
plt.plot(dimensions, validation_accuracies_dim, label='Validation Accuracy', marker
plt.xlabel('Number of Dimensions')
plt.ylabel('Accuracy')
```

```python
plt.title('Effect of Dimensionality Reduction on Accuracy')
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig('dimensionality_reduction.png')
plt.show()

# Process test set: TF-IDF
test_tfidf_features = tfidf_vectorizer.transform(test_data['body'])  # Use the same
test_reduced_tfidf = svd.transform(test_tfidf_features)  # Use the same SVD transfo
test_normalized_tfidf = tfidf_normalizer.transform(test_reduced_tfidf)  # Normalize

# Process test set: Sentence Embeddings
test_sentence_embeddings = sentence_model.encode(test_data['body'].tolist())
test_normalized_sentences = sentence_normalizer.transform(test_sentence_embeddings)

# Combine test features
test_combined = np.hstack([
    test_normalized_tfidf,              # Normalized TF-IDF features
    test_normalized_sentences           # Normalized Sentence Embeddings
])

# Validate feature consistency
print(f"Training features shape: {X_combined.shape}")
print(f"Test features shape: {test_combined.shape}")

# Predict on test set using the best Logistic Regression model
lr_best_model = LogisticRegression(max_iter=3000, C=grid_search.best_params_['C'],
lr_best_model.fit(X_combined, y)
test_predictions = lr_best_model.predict(test_combined)

# Map predictions back to labels
reverse_label_map = {idx: label for label, idx in label_map.items()}
test_data['subreddit'] = [reverse_label_map[int(pred)] for pred in test_predictions

# Create submission file
submission = test_data[['id', 'subreddit']]
submission.to_csv(output_file, index=False)
print(f"Submission file saved as: {output_file}")

# Result Output
results_df.to_csv('results_summary.csv', index=False)
print("Results saved to results_summary.csv")
```

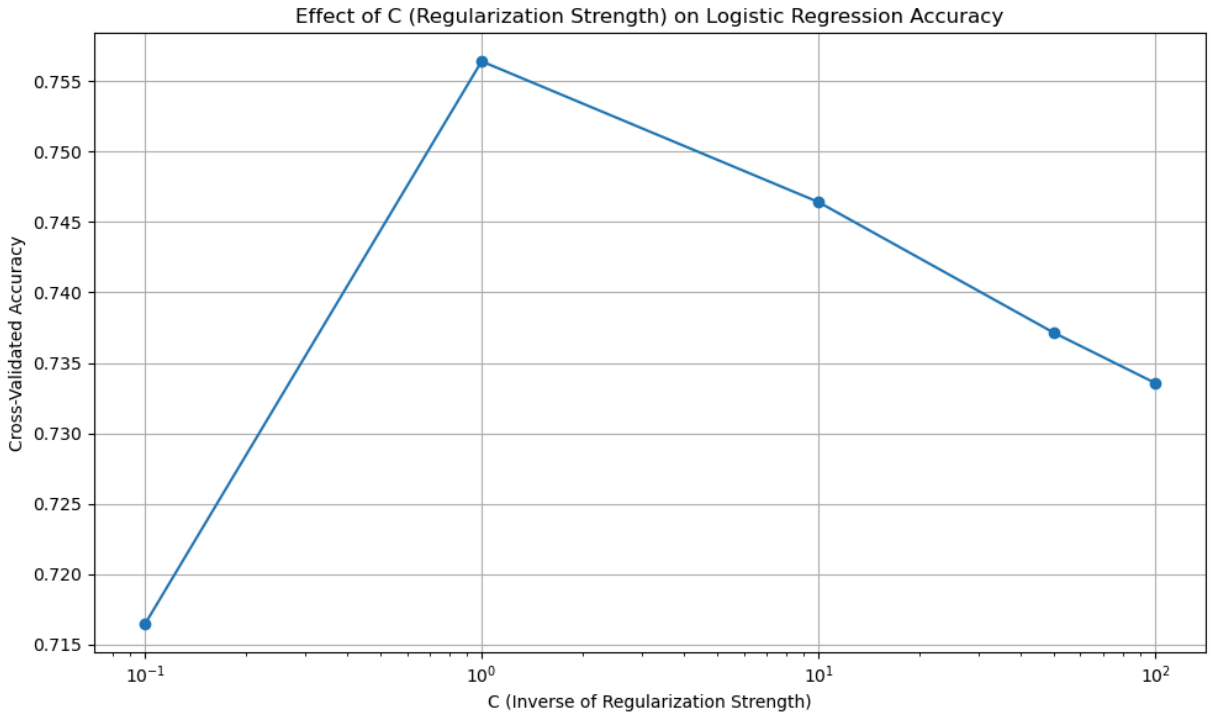
```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Fitting 5 folds for each of 4 candidates, totalling 20 fits
                     Model  Training Accuracy  \
0  Multinomial Naive Bayes           0.619286
1      Logistic Regression           0.756429

                    Validation Accuracy (Class-wise)  Time (5-fold)  \
0  Toronto: 0.6152\nMontreal: 0.6655\nLondon: 0.5...           0.19
1  Toronto: 0.7713\nMontreal: 0.7593\nLondon: 0.7...           1.93

   Numb. Params
0         400.0
1           NaN
```

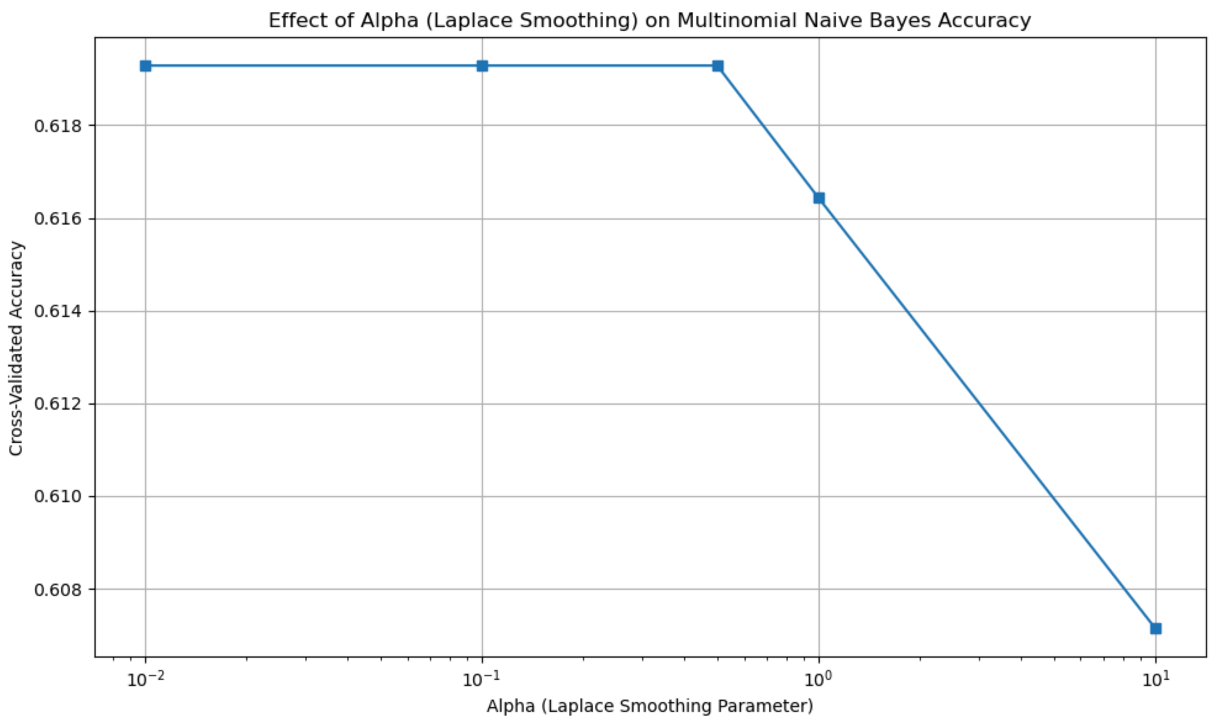




```
Training features shape: (1400, 784)
Test features shape: (600, 784)
Submission file saved as: submissions.csv
Results saved to results_summary.csv
```

In [21]:
```python
import matplotlib.pyplot as plt

# Plot hyperparameter tuning results for Logistic Regression (example: C vs. accura
lr_c_values = param_grid_lr['C']
lr_accuracies = []

for c in lr_c_values:
    model = LogisticRegression(max_iter=3000, C=c, solver='lbfgs')
    scores = cross_val_score(model, X_combined, y, cv=kf, scoring='accuracy')
    lr_accuracies.append(scores.mean())

plt.figure(figsize=(10, 6))
plt.plot(lr_c_values, lr_accuracies, marker='o')
plt.title('Effect of C (Regularization Strength) on Logistic Regression Accuracy')
plt.xlabel('C (Inverse of Regularization Strength)')
plt.ylabel('Cross-Validated Accuracy')
plt.xscale('log')  # C values are better represented on a log scale
plt.grid()
plt.tight_layout()
plt.savefig('logistic_regression_hyperparameter_tuning.png')
plt.show()

# Plot hyperparameter tuning results for Multinomial Naive Bayes (example: alpha vs
nb_alpha_values = param_grid_nb['alpha']
nb_accuracies = []

for alpha in nb_alpha_values:
    model = MultinomialNB(alpha=alpha)
    scores = cross_val_score(model, non_negative_tfidf, y, cv=kf, scoring='accuracy
    nb_accuracies.append(scores.mean())

plt.figure(figsize=(10, 6))
plt.plot(nb_alpha_values, nb_accuracies, marker='s')
plt.title('Effect of Alpha (Laplace Smoothing) on Multinomial Naive Bayes Accuracy'
plt.xlabel('Alpha (Laplace Smoothing Parameter)')
plt.ylabel('Cross-Validated Accuracy')
plt.xscale('log')  # Alpha values are better represented on a log scale
plt.grid()
plt.tight_layout()
plt.savefig('mnb_hyperparameter_tuning.png')
plt.show()
```


Effect of C (Regularization Strength) on Logistic Regression Accuracy
0.755
0.750
0.745
0.740
0.735
0.730
0.725
0.720
0.715
Cross-Validated Accuracy
10^{-1}
10^{0}
10^{1}
10^{2}
C (Inverse of Regularization Strength)


Effect of Alpha (Laplace Smoothing) on Multinomial Naive Bayes Accuracy
0.618
0.616
0.614
0.612
0.610
0.608
Cross-Validated Accuracy
10^{-2}
10^{-1}
10^{0}
10^{1}
Alpha (Laplace Smoothing Parameter)

In [ ]: