# Annexe C: Optimisation d’hyper-paramètres pour les méthodes exploratoires

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import StackingClassifier, VotingClassifier
from sklearn.metrics import make_scorer, f1_score
from sklearn.pipeline import make_pipeline
from lightgbm import LGBMClassifier

# Load and prepare 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)
labels_train_df = pd.read_csv('label_train.csv')
labels_train = labels_train_df['label'].values
df_train = pd.DataFrame(data_train, columns=vocab_map)
df_train['TARGETT'] = labels_train
X = df_train.drop(columns=['TARGETT'])
y = df_train['TARGETT']

# Apply TF-IDF Transformation
print("Applying TF-IDF Transformation...")
tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X)
print("Transformation applied!")

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

# Dimensionality reduction configurations
# Truncated SVD is used for dimensionality reduction in high-dimensional sparse␣
↪data
# SVD 5000 is used to reduce the number of features to 5000
dim_reduction_configs = {
    'SVD_5000': TruncatedSVD(n_components=5000, random_state=42)
```

```
}

# Define models optimized for sparse, high-dimensional data
standard_models = {

    # Logistic Regression using saga solver with elasticnet regularization
    # Saga solver is optimized for large datasets and supports L1 and L2␣
↪regularization
    # Why elasticnet? It combines L1 and L2 regularization, which can be useful␣
↪when there are correlated features
    # Why logistic regression? It's a simple and fast linear model that can work␣
↪well with high-dimensional data
    'LogisticRegression': make_pipeline(
        StandardScaler(with_mean=False),
        LogisticRegression(solver='saga', max_iter=5000, random_state=42,␣
↪penalty='elasticnet', l1_ratio=0.5, n_jobs=-1)
    ),

    # Linear Support Vector Classifier (LinearSVC) with linear kernel
    # Why LinearSVC? It's optimized for large datasets and can handle␣
↪high-dimensional data
    # Why linear kernel? It's suitable for linearly separable data, which is␣
↪common in text classification
    'LinearSVC': make_pipeline(
        StandardScaler(with_mean=False),
        LinearSVC(max_iter=10000, tol=1e-4, penalty='l2', C=0.1, dual=False,␣
↪random_state=42)  # Increased max_iter, adjusted C
    )
}

# Stacking and Voting classifiers
# Stacking: Combines multiple models to improve performance
stacking_model = StackingClassifier(
    # Estimators used:
    # - Logistic Regression : Simple linear model that can work well with␣
↪high-dimensional data
    # - SGD Classifier : Stochastic Gradient Descent Classifier
    # - LinearSVC : Linear Support Vector Classifier
    # - LightGBM Classifier : Gradient boosting model that can handle␣
↪high-dimensional data
    estimators=[
        ('logreg', make_pipeline(StandardScaler(with_mean=False),␣
↪LogisticRegression(solver='saga', max_iter=5000, penalty='elasticnet',␣
↪l1_ratio=0.5, random_state=42))),
```

```
        ('sgd', make_pipeline(StandardScaler(with_mean=False),␣
↪SGDClassifier(loss='hinge', max_iter=5000, tol=1e-3, penalty='elasticnet',␣
↪l1_ratio=0.5, random_state=42))),
        ('svc', make_pipeline(StandardScaler(with_mean=False),␣
↪LinearSVC(max_iter=10000, tol=1e-4, penalty='l2', C=0.1, dual=False,␣
↪random_state=42))),
        ('lgbm', LGBMClassifier(n_estimators=100, learning_rate=0.1,␣
↪num_leaves=15, max_depth=6, random_state=42))
    ],

    # Final estimator is used to combine the predictions of the base estimators␣
↪(default is Logistic Regression)
    # We can also use a different model as the final estimator (e.g., Random␣
↪Forest, Gradient Boosting, etc.)
    final_estimator=LogisticRegression(),
    cv=5, n_jobs=-1
)

# Voting: Combines multiple models by averaging or taking the majority vote
voting_model = VotingClassifier(

    # Estimators used:
    # - Logistic Regression : Simple linear model that can work well with␣
↪high-dimensional data
    # - LinearSVC : Linear Support Vector Classifier
    # - LightGBM Classifier : Gradient boosting model that can handle␣
↪high-dimensional data
    estimators=[
        ('logreg', make_pipeline(StandardScaler(with_mean=False),␣
↪LogisticRegression(max_iter=1000, random_state=42))),
        ('sgd', make_pipeline(StandardScaler(with_mean=False),␣
↪SGDClassifier(loss='hinge', max_iter=5000, tol=1e-3, penalty='elasticnet',␣
↪l1_ratio=0.5, random_state=42))),
        ('svc', SVC(kernel="linear", C=1.0, probability=True, max_iter=20000,␣
↪random_state=42)),  # Increased max_iter
        ('lgbm', LGBMClassifier(n_estimators=100, learning_rate=0.1,␣
↪num_leaves=15, max_depth=6, random_state=42))
    ],

    # Voting method: 'soft' for averaging predicted probabilities, 'hard' for␣
↪majority vote
    voting='soft', n_jobs=-1
)

# Combine models for evaluation
models = {**standard_models, 'Stacking': stacking_model, 'Voting': voting_model}
```

```
results = []

# Apply Chi-Squared feature selection and dimensionality reduction, then␣
↪evaluate models
for feat_name, feature_select in {'ChiSquare_1000': SelectKBest(chi2, k=1000),␣
↪'ChiSquare_1500': SelectKBest(chi2, k=1500)}.items():

    # Chi-Squared feature selection is used to select the most relevant features␣
↪based on the chi-squared statistic
    # It's commonly used for text classification to select features that are␣
↪likely to be related to the target class
    print(f"Applying Chi-Squared feature selection: {feat_name}")
    X_chi2 = feature_select.fit_transform(X_tfidf, y)

    for dim_name, dim_reduction in dim_reduction_configs.items():
        X_reduced = dim_reduction.fit_transform(X_chi2)

        for model_name, model in models.items():
            # Perform cross-validation with F1 scoring and 5-fold stratified␣
↪cross-validation
            # 5-fold instead of 10-fold to reduce computation time
            cv_score = cross_val_score(model, X_reduced, y, scoring=f1_scorer,␣
↪cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42)).mean()
            results.append({
                'Dimensionality Reduction': dim_name,
                'Feature Selection': feat_name,
                'Model': model_name,
                'Best C': None,
                'F1 Score': cv_score
            })
            print(f"Completed: DimReduction={dim_name},␣
↪FeatSelection={feat_name}, Model={model_name}, F1 Score={cv_score:.4f}")

# Save results to CSV
results_df = pd.DataFrame(results)
results_df.to_csv('model_evaluation_results.csv', index=False)
print("All evaluations completed. Results saved to model_evaluation_results.csv")
```

[58]:
```
print('''Dime Red        Feat. Selec.        Model                                    ␣
↪Best C        F1 Score
SVD_300            ChiSquare_1000        LogisticRegression        0.696590847
SVD_300            ChiSquare_1000        SGDClassifier                  0.
↪678114328
SVD_300            ChiSquare_1000        LinearSVC                  0.693165985
SVD_300            ChiSquare_1000        Stacking                  0.684403284
```

```
SVD_300           ChiSquare_1000       Voting                          0.
↪685068626
SVD_1000    ChiSquare_1000        LogisticRegression       0.696230131
SVD_1000    ChiSquare_1000        SGDClassifier                 0.689321683
SVD_1000    ChiSquare_1000        LinearSVC                 0.704406554
SVD_1000    ChiSquare_1000        Stacking                0.737203681
SVD_1000    ChiSquare_1000        Voting                        0.718489279
SVD_300            ChiSquare_1500        LogisticRegression       0.692465761
SVD_300            ChiSquare_1500        SGDClassifier                 0.
↪657953274
SVD_300            ChiSquare_1500        LinearSVC                 0.690176841
SVD_300            ChiSquare_1500        Stacking                0.680429205
SVD_300            ChiSquare_1500        Voting                          0.
↪684510991
SVD_1000    ChiSquare_1500        LogisticRegression       0.707492076
SVD_1000    ChiSquare_1500        SGDClassifier                 0.685584959
SVD_1000    ChiSquare_1500        LinearSVC                0.711376559
SVD_1000    ChiSquare_1500        Stacking                0.74379305   ␣
↪***BEST***
SVD_1000    ChiSquare_1500        Voting                0.725974123

''')
```

```
Dime Red        Feat. Selec.     Model                       Best C    F1 Score
SVD_300      ChiSquare_1000      LogisticRegression       0.696590847
SVD_300      ChiSquare_1000      SGDClassifier            0.678114328
SVD_300      ChiSquare_1000      LinearSVC                0.693165985
SVD_300      ChiSquare_1000      Stacking                 0.684403284
SVD_300     ChiSquare_1000       Voting                   0.685068626
SVD_1000    ChiSquare_1000       LogisticRegression       0.696230131
SVD_1000    ChiSquare_1000       SGDClassifier            0.689321683
SVD_1000    ChiSquare_1000       LinearSVC                0.704406554
SVD_1000    ChiSquare_1000       Stacking                 0.737203681
SVD_1000    ChiSquare_1000       Voting                   0.718489279
SVD_300      ChiSquare_1500      LogisticRegression       0.692465761
SVD_300      ChiSquare_1500      SGDClassifier            0.657953274
SVD_300      ChiSquare_1500      LinearSVC                0.690176841
SVD_300      ChiSquare_1500      Stacking                 0.680429205
SVD_300      ChiSquare_1500      Voting                   0.684510991
SVD_1000    ChiSquare_1500       LogisticRegression       0.707492076
SVD_1000    ChiSquare_1500       SGDClassifier            0.685584959
SVD_1000    ChiSquare_1500       LinearSVC                0.711376559
SVD_1000    ChiSquare_1500       Stacking                 0.74379305 **BEST**
SVD_1000    ChiSquare_1500       Voting                   0.725974123
```