What was the highest performing pipeline and its corresponding ROC AUC score among the evaluated pipelines?

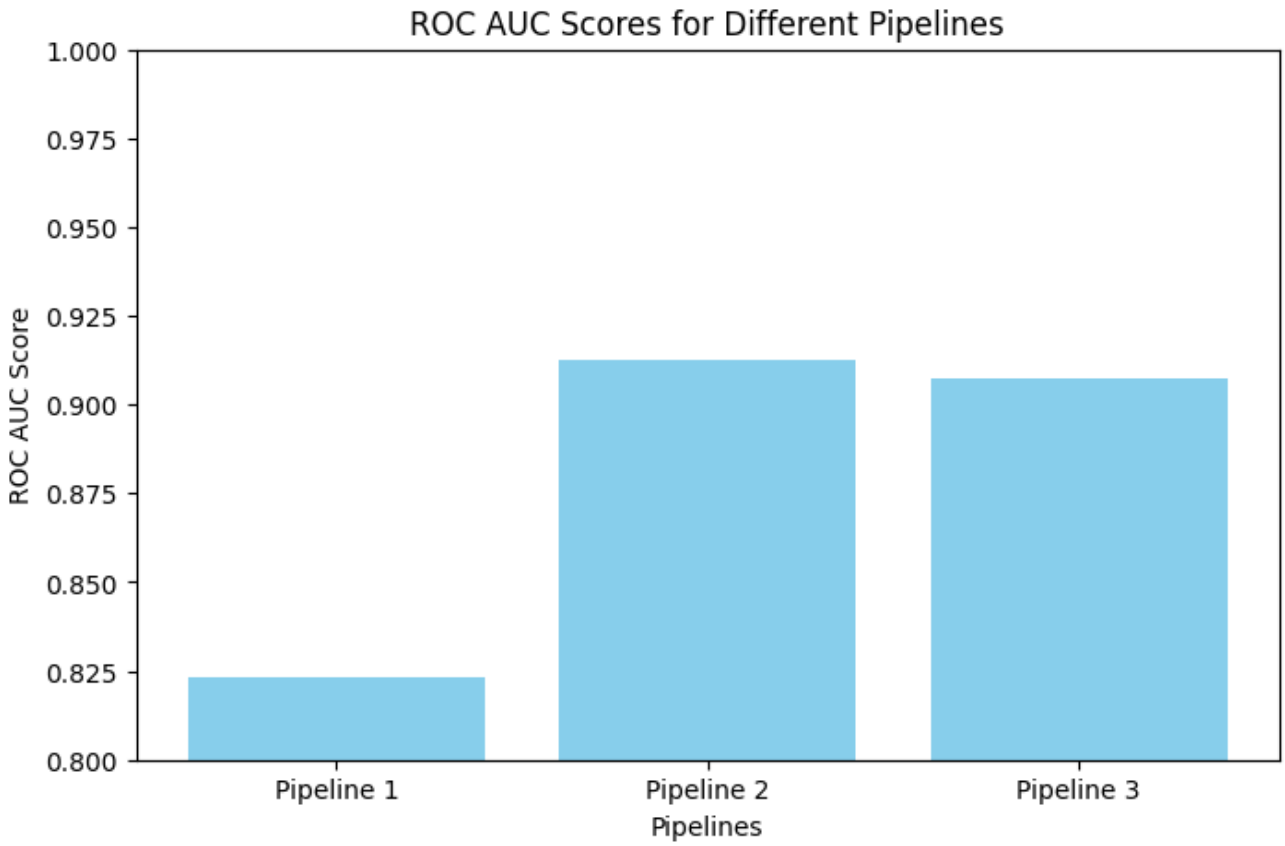
After evaluating the three pipelines on a smaller subset of the dataset, Pipeline 2 emerged as the preferred choice for the final pipeline. While Pipeline 1 achieved a respectable ROC AUC score of 0.8230, both Pipeline 2 and Pipeline 3 outperformed it significantly, with ROC AUC scores of 0.9127 and 0.9076, respectively. Despite Pipeline 3's incorporation of both tfidf and engineered features, Pipeline 2, which solely utilizes engineered features, demonstrated comparable performance. Moreover, Pipeline 2 offers the advantage of being considerably faster than Pipeline 3 due to its simpler feature set. Therefore, based on the analysis of model scores and efficiency, Pipeline 2 is selected as the final pipeline for further training and deployment.

```
'''
+------------+--------------------+
| Pipeline   |   AUC Score (test) |
|------------|--------------------|
| Pipeline 1 |      0.8230        |
| Pipeline 2 |      0.9127        |
| Pipeline 3 |      0.9076        |
+------------+--------------------+
'''
```

Running the best pipeline on 40% sampled data

```
import matplotlib.pyplot as plt

# Pipeline names
pipelines = ['Pipeline 1', 'Pipeline 2', 'Pipeline 3']

# ROC AUC scores
roc_auc_scores = [0.8230, 0.9127, 0.9076]

# Plotting ROC AUC scores
plt.figure(figsize=(8, 5))
plt.bar(pipelines, roc_auc_scores, color='skyblue')
plt.title('ROC AUC Scores for Different Pipelines')
plt.xlabel('Pipelines')
```

```
14 plt.ylabel('ROC AUC Score on test data')
15 plt.ylim(0.8, 1.0)  # Limiting y-axis for better visualization
16 plt.show()
```




```
1 #Load Dataset
2 import pandas as pd
3 df = pd.read_csv('spam.csv', encoding='ISO-8859-1')
4 df.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already	NaN	NaN	NaN

Next steps: Generate code with df View recommended plots

```
1 # Rename columns
2 df = df[['v1', 'v2']]
```

```
df = df[['v1', 'v2']]
df.columns = ['label', 'message']
df.head()
```

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Next steps: Generate code with df View recommended plots

```
#Convert to Binary if spam then 1, else 0 to process ahead
df['label'] = df['label'].map(lambda x: 1 if x == 'spam' else 0) # converting
```

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split
from plot_learning_curve import plot_learning_curve
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.compose import ColumnTransformer
from FeaturizerSpacy import ManualFeatures
from sklearn.pipeline import Pipeline
import CustomPreprocessorSpacy as cp
from scipy.sparse import csr_matrix
from xgboost import XGBClassifier
from pathlib import Path
import numpy as np
import joblib
import matplotlib.pyplot as plt
```

```
#Large subset sampling
df_ = df.sample(frac=0.4, replace=True, random_state=1)
```

```
#Extracting values of column message into X and label into Y, as numpy arrays
X, y = df_['message'].values, df_['label'].values
```

```
base_folder = Path('./')
data_folder = base_folder/'datasets/'
model_folder = base_folder/'models/'
```

```
4
5 model_folder.mkdir(exist_ok=True, parents=True)
6 data_folder.mkdir(exist_ok=True, parents=True)
```

```
1 # Splitting the data into train test splits with test as 20%
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

```
1 # Cleaning the text data in X_train and X_test
2 X_train_cleaned_basic = cp.SpacyPreprocessor(model='en_core_web_sm',
3                                               lemmatize=False, lower=False,
4                                               remove_stop=False, remove_punct=F
5                                               remove_email=False, remove_url=Fa
6                                               add_user_mention_prefix=False,
7                                               basic_clean_only=True).transform(
8
9 X_test_cleaned_basic = cp.SpacyPreprocessor(model='en_core_web_sm',
10                                              lemmatize=False, lower=False,
11                                              remove_stop=False, remove_punct=Fa
12                                              remove_email=False, remove_url=Fal
13                                              add_user_mention_prefix=False,
14                                              basic_clean_only=True).transform(X
15
16 # save  this to a file
17 file_X_train_cleaned_basic = data_folder / 'x_train_cleaned_basic_small.pkl'
18 joblib.dump(X_train_cleaned_basic, file_X_train_cleaned_basic)
19
20 # save  this to a file
21 file_X_test_cleaned_basic = data_folder / 'x_test_cleaned_basic_small.pkl'
22 joblib.dump(X_test_cleaned_basic, file_X_test_cleaned_basic)
```

```
/content/CustomPreprocessorSpacy.py:83: MarkupResemblesLocatorWarning: The in
  soup = BeautifulSoup(text, "html.parser")
['datasets/x_test_cleaned_basic_small.pkl']
```

```
1 #Using manual features
2 featurizer = ManualFeatures(spacy_model='en_core_web_sm', batch_size=1000)
3 X_train_cleaned_basic = joblib.load(file_X_train_cleaned_basic)
4 X_train_features, feature_names = featurizer.fit_transform(X_train_cleaned_bas
```

```
1 pd.DataFrame(X_train_features, columns=feature_names).head()
```

	count_words	count_characters	count_characters_no_space	avg_word_length
0	7.0	35.0	29.0	3.625000
1	14.0	67.0	55.0	3.666667
2	15.0	76.0	63.0	3.937500
3	20.0	88.0	70.0	3.333333
4	24.0	149.0	126.0	5.040000

```
1 #defining classifier and params grid
2 classifier_final = Pipeline([('classifier', XGBClassifier())])
3 params_classifier_final = {'classifier__n_estimators': [200, 400, 600, 800],
4                        'classifier__max_depth': [1, 2, 3, 4]}
```

```
1 #Defining randomsearch CV
2 randomsearchCV_classifier_final = RandomizedSearchCV(estimator=classifier_fina
```

```
1 #Fitting RandomSearchCV on train data
2 randomsearchCV_classifier_final.fit(X_train_features, y_train)
```

▸ **RandomizedSearchCV**

▸ **estimator: Pipeline**

▸ XGBClassifier

```
1 ## let's check the cross validation score
2 print(f'Best cross-validation score: {randomsearchCV_classifier_final.best_sco
3 print("\nBest parameters: ", randomsearchCV_classifier_final.best_params_)
4 print("\nBest estimator: ", randomsearchCV_classifier_final.best_estimator_)
```

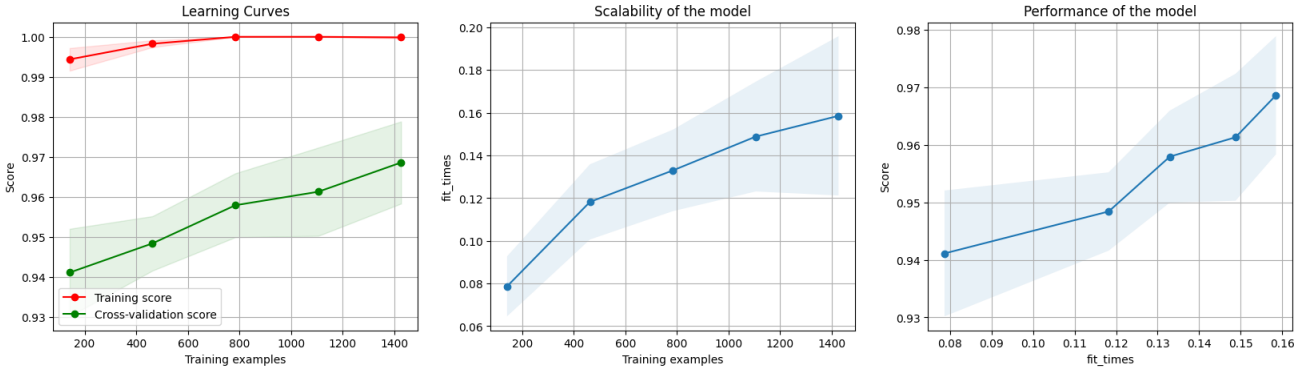
```
Best cross-validation score: 0.98

Best parameters:  {'classifier__n_estimators': 200, 'classifier__max_depth': 1

Best estimator:  Pipeline(steps=[('classifier',
                 XGBClassifier(base_score=None, booster=None, callbacks=None,
                               colsample_bylevel=None, colsample_bynode=None,
                               colsample_bytree=None, device=None,
                               early_stopping_rounds=None,
                               enable_categorical=False, eval_metric=None,
                               feature_types=None, gamma=None, grow_policy=Non
                               importance_type=None,
                               interaction_constraints=None, learning_rate=Non
                               max_bin=None, max_cat_threshold=None,
                               max_cat_to_onehot=None, max_delta_step=None,
                               max_depth=3, max_leaves=None,
                               min_child_weight=None, missing=nan,
                               monotone_constraints=None, multi_strategy=None,
                               n_estimators=200, n_jobs=None,
                               num_parallel_tree=None, random_state=None, ...
```

```
1 file_best_estimator_pipeline_final = model_folder / \
2     'pipeline_final_best_estimator.pkl'
3 file_complete_grid_pipeline_final = model_folder / \
4     'pipeline_final_complete_grid.pkl'
5
6 joblib.dump(randomsearchCV_classifier_final.best_estimator
```

```
joblib.dump(randomsearchCV_classifier_final.best_estimator_,
            file_best_estimator_pipeline_final)
joblib.dump(randomsearchCV_classifier_final, file_complete_grid_pipeline_final

# load the saved model
best_estimator_pipeline_final = joblib.load(file_best_estimator_pipeline_final
complete_grid_pipeline_final = joblib.load(file_complete_grid_pipeline_final)
```

```
#Learning curves
%%time
plot_learning_curve(best_estimator_pipeline_final, 'Learning Curves',
                    X_train_features, y_train, n_jobs=-1)
```

```
CPU times: user 193 ms, sys: 61.3 ms, total: 254 ms
Wall time: 3.88 s
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/
matplotlib/pyplot.py'>
```




```
# let's check the train scores
print(best_estimator_pipeline_final.score(X_train_features, y_train))

# let's check the cross validation score
print(complete_grid_pipeline_final.best_score_)
```

```
0.9994391475042064
0.9814274328094077
```

```
# Final Pipeline
def final_pipeline(text):
    text_cleaned = joblib.load(file_X_test_cleaned_basic)
    features, feature_names = featurizer.fit_transform(text_cleaned)
```

```
    best_estimator_pipeline_final = joblib.load(
        file_best_estimator_pipeline_final)
    predictions = best_estimator_pipeline_final.predict(features)
    return predictions
```

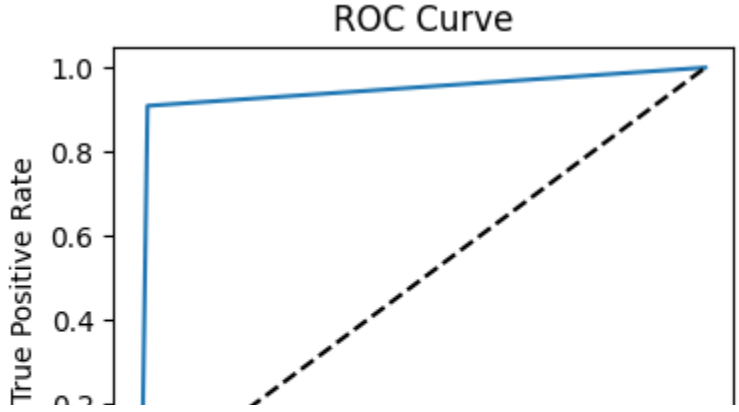
```
#Predicting on test values
y_test_pred = final_pipeline(X_test)
```

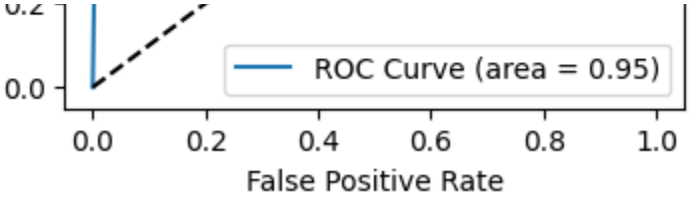
```
#Classification report on test data
print('\nTest set classification report:\n\n', classification_report(y_test, y
```

```
Test set classification report:

                precision    recall  f1-score   support

           0       0.99      0.99      0.99       391
           1       0.93      0.91      0.92        55

    accuracy                           0.98       446
   macro avg       0.96      0.95      0.95       446
weighted avg       0.98      0.98      0.98       446
```

```
#ROC Score
roc_auc = roc_auc_score(y_test, y_test_pred)
print(f"ROC AUC Score: {roc_auc}")

fpr, tpr, thresholds = roc_curve(y_test, y_test_pred)

# Plot ROC curve
plt.figure(figsize=(4,3))
plt.plot(fpr, tpr, label=f'ROC Curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

```
ROC AUC Score: 0.9494303650313881
```





1

The final pipeline, trained and tested on 40% of the dataset, achieved an impressive ROC AUC score of 0.9494, indicating robust performance in classifying the data.

1 Start coding or generate with AI.