Sai Subhash M
BL.EN.U4AIE21113

Amrita Vishwa Vidyapeetham, Bangalore Campus

Kaiburr Assessment Submission

Taks-6

Task-6 Data Science example

First import necessary libraries for the implementation

1. Explanatory Data Analysis and Feature Engineering

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dataset/complaints/complaints.csv')
df.shape
```

```
<ipython-input-7-d9659a6da6b6>:1: DtypeWarning: Columns (16) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dataset/complaints/complaints.csv')
```

```
(4091495, 18)
```

```
df.head()
```

	Date received	Product	Sub-product	Issue	Sub-issue	Consumer complaint narrative	Company public response	Company	State	ZIP code	Tags	Consumer consent provided?	Submitted via	Da sent compai
0	2023-08-24	Credit reporting, credit repair services, or o...	Credit reporting	Problem with a credit reporting company's inve...	Was not notified of investigation status or re...	NaN	NaN	Experian Information Solutions Inc.	NJ	07024	NaN	Other	Web	2023-0
1	2023-08-25	Credit reporting or other personal consumer re...	Credit reporting	Improper use of your report	Reporting company used your report improperly	NaN	NaN	SANTANDER HOLDINGS USA, INC.	FL	33972	NaN	NaN	Web	2023-0
2	2023-07-13	Checking or savings account	Checking account	Problem caused by your funds being low	Overdrafts and overdraft fees	Citibank allowed debit card transactions to ov...	Company has responded to the consumer and the ...	CITIBANK, N.A.	TX	XXXXX	NaN	Consent provided	Web	2023-0
3	2023-08-25	Credit reporting or other personal consumer re...	Credit reporting	Improper use of your report	Reporting company used your report improperly	NaN	NaN	EQUIFAX, INC.	FL	33884	Servicemember	NaN	Web	2023-0
4	2023-09-13	Credit reporting or other personal consumer re...	Credit reporting	Problem with a company's investigation into an...	Their investigation did not fix an error on yo...	NaN	NaN	SANTANDER HOLDINGS USA, INC.	TX	77521	NaN	NaN	Web	2023-0

Fig. 1, First 4 rows & columns will be displayed from the dataset

```
df.columns
```

```
Index(['Date received', 'Product', 'Sub-product', 'Issue', 'Sub-issue',
       'Consumer complaint narrative', 'Company public response', 'Company',
       'State', 'ZIP code', 'Tags', 'Consumer consent provided?',
       'Submitted via', 'Date sent to company', 'Company response to consumer',
       'Timely response?', 'Consumer disputed?', 'Complaint ID'],
      dtype='object')
```

Fig. 2, Columns titled will be displayed

```
df.isnull().sum()
# Total Rows 4091495
```

```
Date received                        0
Product                              0
Sub-product                     235291
Issue                                0
Sub-issue                       716727
Consumer complaint narrative   2608933
Company public response        2196051
Company                              0
State                            43234
ZIP code                         30213
Tags                           3664664
Consumer consent provided?      901658
Submitted via                        0
Date sent to company                 0
Company response to consumer         5
Timely response?                     0
Consumer disputed?             3323137
Complaint ID                         0
dtype: int64
```

```
og_df_len = len(df)
```

Fig.3, To Perform a Text Classification on this dataset we only need 'Product' and 'Consumer complaint narrative' columns. So removed the other columns.

```
df.head()
```

	Product	Complaint
2	Checking or savings account	Citibank allowed debit card transactions to ov...
6	Credit reporting, credit repair services, or o...	I submitted a letter to the XXXX Credit Bureau...
37	Credit reporting, credit repair services, or o...	In accordance with the Fair Credit Reporting a...
38	Credit reporting, credit repair services, or o...	On XX/XX/, 2023, XXXX XXXX admitted liability ...
40	Credit reporting, credit repair services, or o...	XX/XX/XXXX] [XXXX XXXX XXXX] [XXXX XXXX XX...

Fig. 4, Products & Complaints written, only 36.24 % of the data are having Complaint Texts.

```
# Renamed the categories for less confusion during predection
df.replace({'Product':
            {'Credit reporting, credit repair services, or other personal consumer reports':
             'Credit reporting, repair, or other',
             'Credit reporting': 'Credit reporting, repair, or other',
             'Credit card': 'Credit card or prepaid card',
             'Prepaid card': 'Credit card or prepaid card',
             'Payday loan': 'Payday loan, title loan, or personal loan',
             'Money transfer': 'Money transfer, virtual currency, or money service',
             'Virtual currency': 'Money transfer, virtual currency, or money service'}},
           inplace= True)
```

Fig. 5, Renamed the categories for better understanding for prediction

Classes used for prediction: -

0. Credit reporting, repair, or other

1. Debt collection

2. Consumer Loan

3. Mortgage

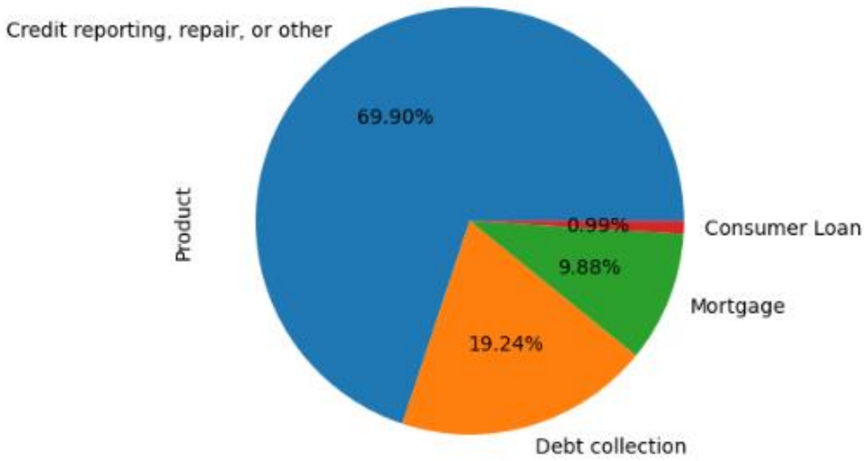


Fig. 6, Visualizing multi-classes distribution in pi-chart

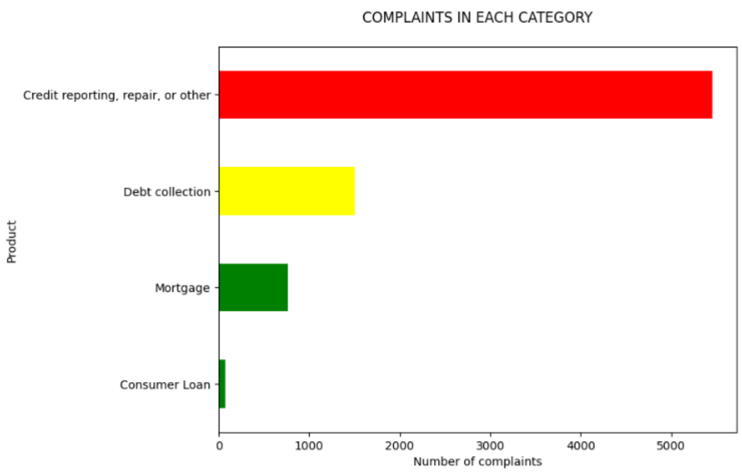


Fig. 7 Complaints in each category for Product vs Number of Complaints

Sai Subhash M
BL.EN.U4AIE21113

2. Text Processing: -

The function clean_text() performs the following text preprocessing steps:

- Converts the text to lowercase. This is important because most NLP algorithms are case-insensitive.
- Tokenizes the text into words. This involves splitting the text into individual words, taking into account punctuation and contractions.
- Removes punctuation and special characters. This is done to reduce the number of features that the NLP algorithm needs to consider.
- Removes numerical digits. This is done because NLP algorithms are typically trained on text data, not numerical data.
- Removes common stop words. Stop words are words that are very common in a language and do not add much meaning to a sentence. Removing stop words can improve the performance of NLP algorithms by reducing the number of features that they need to consider.
- (Optional) Lemmatizes words to their base form. Lemmatization is the process of converting words to their base form. For example, the words "walk," "walking," and "walker" would all be lemmatized to the word "walk." Lemmatization can improve the performance of NLP algorithms by making it easier for them to understand the meaning of words.

pre_process_texts()

The steps performed by the pre_process_texts() function:

- Clean the text data. This involves removing punctuation, stop words, and other noise.
- Lemmatize the text data (optional). This involves converting words to their base form.
- Vectorize the text data. This involves converting the text data to a numerical representation that can be used by machine learning algorithms.

```
df['Complaint'].iloc[1]
```

```
"My house was broken into several years ago and I've been having problems with ppl using my ID to obtain things for years. Pls make this all go away!"
```

```
df['Complaint'] = df['Complaint'].apply(lambda x: clean_text(x))
df['Complaint'].iloc[1]
```

```
'house broken several year ago problem ppl using id obtain thing year pls make go away'
```

- The texts are now clean enough for proceeding, For feature engineering, I am using TF-IDF, which is a common technique for text classification.
- TF-IDFs are word frequency scores that try to highlight words that are more interesting.

```
[28]: #TF-IDFs are word frequency scores that try to highlight words that are more interesting.
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5,
                        ngram_range=(1, 2),
                        stop_words='english')

# Transform each complaint into a vector
features = tfidf.fit_transform(df.Complaint).toarray()

labels = df.encoded_label
```

```
[31]: # Three most correlated terms with each of the 4 product categories
N = 3
for Product, category_id in sorted(category_map.items()):
  features_chi2 = chi2(features, labels == category_id)
  indices = np.argsort(features_chi2[0])
  feature_names = np.array(tfidf.get_feature_names_out())[indices]
  unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
  bigrams = [v for v in feature_names if len(v.split(' ')) == 2]
  print("\n%s:" %(Product))
  print("______________________________________")
  print("  - Most Correlated Unigrams:    %s" %(', '.join(unigrams[-N:])))
  print("  - Most Correlated Bigrams:     %s" %(', '.join(bigrams[-N:])))
```

```
Consumer Loan:
_________________________________________
  - Most Correlated Unigrams:    ally, toyota, car
  - Most Correlated Bigrams:     car time, repossessed car, purchased vehicle

Credit reporting, repair, or other:
_________________________________________
  - Most Correlated Unigrams:    section, mortgage, debt
  - Most Correlated Bigrams:     reporting agency, right privacy, section state

Debt collection:
_________________________________________
  - Most Correlated Unigrams:    owe, collection, debt
  - Most Correlated Bigrams:     collect debt, debt collection, collection agency

Mortgage:
_________________________________________
  - Most Correlated Unigrams:    escrow, modification, mortgage
  - Most Correlated Bigrams:     mortgage company, mortgage payment, loan modification
```

3. Selection of models for multi-class classification: -

I. Linear Support Vector Machine (LinearSVM)
II. Logistic Regression
III. Random Forest
IV. Multinomial Naive Bayes
V. K Neighbors Classifier
VI. AdaBoost Classifier
VII. Bagging Classifier

```
models = [
    RandomForestClassifier(n_estimators=100, max_depth=5, random_state=0),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(random_state=0),
    KNeighborsClassifier(),
    AdaBoostClassifier(),
    BaggingClassifier()
]

# 5 Cross-validation
CV = 5
crossval_df = pd.DataFrame(index=range(CV * len(models)))

entries = []
for model in models:
  model_name = model.__class__.__name__
  print(f"Current Model : {model_name}")
  accuracies = cross_val_score(model, features, labels, scoring='accuracy', cv=CV)
  for fold_idx, accuracy in enumerate(accuracies):
    entries.append((model_name, fold_idx, accuracy))

crossval_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'accuracy'])
```

4. Comparison of Model Performance

For comparison I used Mean and Standard Deviation, which are the easiest and most popular parameters for model comparison.

```
mean = crossval_df.groupby('model_name').accuracy.mean()
std = crossval_df.groupby('model_name').accuracy.std()

acc = pd.concat([mean, std], axis= 1,
          ignore_index=True)
acc.columns = ['Mean Accuracy', 'Standard deviation']
acc
```

model_name	Mean Accuracy	Standard deviation
AdaBoostClassifier	0.831731	0.007152
BaggingClassifier	0.861998	0.003345
KNeighborsClassifier	0.854048	0.005593
LinearSVC	0.896627	0.004115
LogisticRegression	0.893933	0.004355
MultinomialNB	0.856355	0.003968
RandomForestClassifier	0.700141	0.000838

Model	Mean Accuracy	Standard Deviation	Description
LinearSVC	0.8966	0.0041	Highest accuracy, low variability, strong candidate.
Logistic Regression	0.8939	0.0044	High accuracy, good consistency, interpretable.
Multinomial Naive Bayes	0.8564	0.004	Simple, interpretable, decent accuracy.
K-Nearest Neighbors (KNN)	0.854	0.0056	Reasonable accuracy, slightly higher variability.
Bagging Classifier	0.8641	0.0024	Good accuracy, low variability, robust ensemble model.
AdaBoost Classifier	0.8316	0.0071	Lower accuracy, higher variability.
Random Forest Classifier	0.7001	0.0008	Lowest accuracy, not ideal for this task.

Table.1, Comparison of models

So, we can clearly see that, **LinearSVC and Logistic Regression** stand out as strong candidates due to their high mean accuracy and relatively low standard deviation.

5. Model evaluation: -

clean_dataset()

The function clean_dataset() performs the following steps to clean and preprocess a dataset:

- Removes rows with missing 'Consumer complaint narrative'.
- Renames columns to 'Product' and 'Complaint'.
- Optionally samples the dataset if sample_size is provided.
- Maps similar product categories to consolidated categories.
- Generates a category mapping (category names to category IDs)

evaluate_prediction()

The function evaluate_prediction() evaluates the performance of a regression model using various metrics by taking the following input parameters:

- predicted_values: A NumPy array containing the predicted values from the model.
- actual_values: A NumPy array containing the actual target values.
- The function returns a dictionary containing the following evaluation metrics:
- Accuracy: The percentage of predictions that are correct.
- Mean Squared Error (MSE): The average squared difference between the predicted and actual values.
- Root Mean Squared Error (RMSE): The square root of the MSE.
- Mean Absolute Error (MAE): The average absolute difference between the predicted and actual values.
- R-squared (R2) Score: A measure of how well the model explains the variation in the data.
- Explained Variance Score: A measure of how well the model predicts the actual values.

```
category_map
```

```
{'Credit reporting, repair, or other': 0,
 'Debt collection': 1,
 'Consumer Loan': 2,
 'Mortgage': 3}
```

```
df.head()
```

	Product	Complaint	encoded_label
3285291	Credit reporting, repair, or other	I have disputed the errors on my credit report...	0
661888	Credit reporting, repair, or other	My house was broken into several years ago and...	0
2379215	Credit reporting, repair, or other	In accordance with the fair credit reporting a...	0
154588	Credit reporting, repair, or other	I sent a letter to TransUnion regarding a publ...	0
992576	Credit reporting, repair, or other	My credit reports are inaccurate. These inaccu...	0

1. High Mean Accuracy: LinearSVC exhibited the highest mean accuracy among all the models we tested, scoring at approximately 89.67%. This implies that it consistently performs well in classifying text data into multiple categories.

2. Low Standard Deviation: LinearSVC also showcased a relatively low standard deviation of around 0.0041. This indicates that its performance is consistent across different cross-validation folds, making it a stable and reliable choice for our task.

3. Robustness: LinearSVC is known for its robustness and effectiveness in handling high-dimensional data, which is often the case with text data. It can effectively separate text data into multiple classes with minimal overfitting.

4. Interpretability: LinearSVC provides good interpretability. It allows us to understand which features (words or terms) are more important in making classification decisions. This interpretability can be valuable for understanding the factors driving classification outcomes.

5. Speed and Scalability: LinearSVC is computationally efficient and can handle large datasets, making it suitable for our task even if we decide to scale up our dataset in the future.

In summary, LinearSVC not only showcased exceptional performance but also provides transparency and dependability, both of which are pivotal aspects in text classification.

```
# Define the parameter grid for tuning
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10],  # Regularization parameter
    'penalty': ['l2'],  # Regularization penalty type
    'max_iter': [1000, 2000, 3000]  # Maximum number of iterations
}

svc = LinearSVC()

# Create GridSearchCV with 5-fold cross-validation
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=5, scoring='accuracy', verbose=1, n_jobs=-1)

grid_search.fit(X_train, y_train)

print("Best Parameters: ", grid_search.best_params_)
print("Best Accuracy: ", grid_search.best_score_)

# Get the best model
best_svc = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
Best Parameters:  {'C': 1, 'max_iter': 1000, 'penalty': 'l2'}
Best Accuracy:  0.8948109058927001
```

Here I used GridSearchCV to tune the hyperparameters of a LinearSVC model, GridSearchCV is a powerful tool that can be used to find the best combination of hyperparameters for a variety of machine learning models.

```
y_pred = best_svc.predict(X_test)
```

```
eval_matric = evaluate_prediction(predicted_values = y_pred, actual_values = y_test)
```

```
for mat, val in eval_matric.items():
  print(f"{mat}   :  {val}")
```

```
Accuracy   :  0.9044854881266491
MSE   :  0.21688654353562006
RMSE   :  0.4657107938792272
MAE   :  0.12823218997361477
R2   :  0.7440431355073291
Explained Variance   :  0.7447086243409196
```

Metric	**Value**
Accuracy	**0.9045**
MSE	0.2169
RMSE	0.4657
MAE	0.1282
R^2 Score	0.744
Explained Variance	0.7447

Tabel.2, Using grid search applied Hyperparameters for LinearSVC

1. Accuracy: The accuracy of approximately 90.45% indicates that the model correctly predicted the category for nearly 90.45% of the test samples. It's a good measure of overall correctness.

2. MSE (Mean Squared Error): With an MSE of approximately 0.217, this metric quantifies the average squared difference between the predicted and actual values. Lower values are better, indicating that the model's predictions are closer to the actual values.

3. RMSE (Root Mean Squared Error): The RMSE of approximately 0.466 is the square root of the MSE and represents the average prediction error. It's useful for understanding the magnitude of errors, with lower values indicating better predictive accuracy.

4. MAE (Mean Absolute Error): The MAE of approximately 0.128 suggests that, on average, the model's predictions differ from the actual values by around 12.8%. It's another measure of prediction accuracy.

5. R2 (R-squared) Score: The R2 score of approximately 0.744 indicates that about 74.4% of the variance in the target variable is explained by the model. This is a good measure of how well the model fits the data, with higher values indicating a better fit.

6. Explained Variance: The explained variance of approximately 0.745 reflects the proportion of variance in the target variable that the model can explain. It's similar to the R2 score and suggests that the model captures a significant portion of the variance.

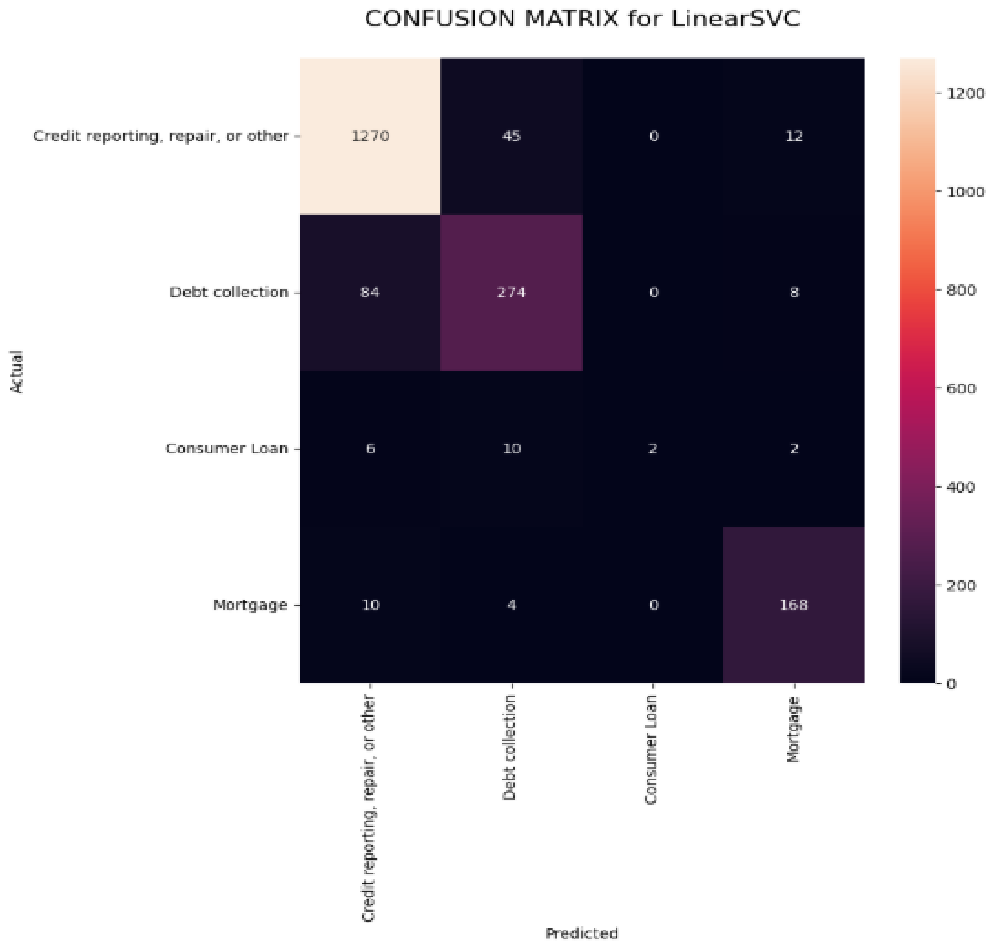
```
print(metrics.classification_report(y_test, y_pred, target_names= categories_to_keep, zero_division=0))
```

```
                                 precision    recall  f1-score   support

Credit reporting, repair, or other       0.93      0.96      0.94      1327
                  Debt collection       0.82      0.75      0.78       366
                    Consumer Loan       1.00      0.10      0.18        20
                         Mortgage       0.88      0.92      0.90       182

                         accuracy                           0.90      1895
                        macro avg       0.91      0.68      0.70      1895
                     weighted avg       0.90      0.90      0.90      1895
```

Classification report for precision, recall, f1-score for multiclasses



Confusion matrix for LinearSVC model Actual vs Predicted

6. Prediction:

predict_categories

The function predict_categories() predicts categories for complaint text data using a trained model. It takes the following input parameters:

a) text: The input text to be predicted.
b) model: The trained classification model.
c) vectorizer: The vectorizer used for text preprocessing.
d) label encoder: The label encoder used for category mapping.

The function returns a string containing the predicted category.

Conclusion:

LinearSVC achieved superior performance over all other tested models thus making it the optimal choice for text classification. Within the cross-validation folds LinearSVC achieves 89.67% mean accuracy with a low standard deviation of 0.0041 which demonstrates its strong and reliable performance.

LinearSVC stands out as a versatile model because it effectively handles text data of high dimensions and features an interpretable algorithm while maintaining scalable performance. Our task requires a model which balances accuracy with stability and computational performance thus LinearSVC stands as the most appropriate choice among the available models.