ML Challenge

CSC311: Intro to Machine Learning

naseerm4, sohailne, kanounad, trckovak

April 8, 2024

naseerm4, sohailne,
kanounad, trckovak

Data Exploration

In our data exploration phase, we analyzed a dataset containing 1468 rows and 10 features, with an additional label indicating the city. During our preliminary assessment, we encountered anomalies such as missing values. To gain a deeper understanding of the dataset, we employed various Pandas functions, notably `.describe()`, to examine the central tendency and dispersion measures (mean, median, mode, count) for numerical features like Q1-Q4. This exploration revealed inconsistencies, as some features had counts less than 1468, indicating missing values. To further investigate, we used `.value_counts()` to examine the distribution of responses, uncovering outliers like an abnormally high 'number of different fashion styles' reported as 1000. For a clearer visual understanding of feature distribution across cities, we employed `boxplot()`, which elucidated the data's spread and variance. Additionally, to analyze qualitative data, specifically Q10 that describes the city, we utilized `crosstab()`, where we found some incomprehensible sentences, further highlighting the need for thorough data cleaning and preprocessing before model development.

Below, you'll find the diagrams showcasing the outputs from various Pandas functions used during our data exploration phase. These visualizations represent the analyses and patterns identified in the dataset, including distributions and anomalies.

Data

	id	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Label
0	42659	4.0	1.0	2.0	1.0	Co-worker	Skyscrapers=>6,Sport=>4,Art and Music=>2,Carni...	25	7.0	100	Slavery	Dubai
1	508149	4.0	3.0	5.0	2.0	Co-worker	Skyscrapers=>6,Sport=>1,Art and Music=>2,Carni...	20	3.0	4	Wherever there is great property, there is gre...	Dubai
2	496935	5.0	4.0	5.0	1.0	Partner,Friends	Skyscrapers=>6,Sport=>2,Art and Music=>3,Carni...	32	5.0	4	Futuristic land	Dubai
3	502824	5.0	4.0	4.0	1.0	Partner,Friends	Skyscrapers=>6,Sport=>5,Art and Music=>1,Carni...	23	10.0	3	The city where anything is possible	Dubai
4	523028	4.0	3.0	3.0	3.0	Partner,Friends,Siblings	Skyscrapers=>6,Sport=>4,Art and Music=>1,Carni...	20	10.0	5	If you can think of a high building, it probab...	Dubai
...	...	...	...	...	...	...	...	...	...	...	...	...
1463	394579	4.0	2.0	5.0	3.0	Partner,Friends,Siblings	Skyscrapers=>2,Sport=>1,Art and Music=>4,Carni...	5	2.0	2	"The average piece of junk is probably more me...	Paris
1464	499389	5.0	3.0	4.0	2.0	Partner,Co-worker	Skyscrapers=>1,Sport=>3,Art and Music=>6,Carni...	7	2.0	4	Oui oui baguette	Paris
1465	522120	2.0	1.0	3.0	4.0	Friends	Skyscrapers=>2,Sport=>3,Art and Music=>6,Carni...	9	87.0	67	E OhÂ	Paris
1466	519777	5.0	4.0	5.0	3.0	Partner,Friends	Skyscrapers=>5,Sport=>6,Art and Music=>2,Carni...	15	1.0	15	croissants and cigarettesÂ	Paris
1467	517373	4.0	3.0	5.0	4.0	Partner	Skyscrapers=>6,Sport=>5,Art and Music=>1,Carni...	5	5.0	5	Secrets travel fast in Paris.	Paris

1468 rows × 12 columns

Figure 1: Reading Data

comments:
An examination of the data that details the total number of columns and rows provides foundational knowledge of the dataset's structure and scale. Understanding the number of columns informs us about the dimensionality of our data, which can highly influence the model design process.

```
[4]  data.describe()
```

	id	Q1	Q2	Q3	Q4	Q8
count	1468.000000	1462.000000	1461.000000	1461.000000	1462.000000	1461.000000
mean	459092.092643	4.321477	3.521561	3.707734	3.391929	4.796030
std	114308.717089	0.884464	1.192031	1.142037	1.297189	22.142305
min	5978.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	406181.000000	4.000000	3.000000	3.000000	2.000000	2.000000
50%	503173.000000	5.000000	4.000000	4.000000	3.000000	3.000000
75%	522169.000000	5.000000	5.000000	5.000000	5.000000	5.000000
max	727099.000000	5.000000	5.000000	5.000000	5.000000	800.000000

Figure 2: Data Description

comments:
`.describe()` helps to examine the central tendency and dispersion measures (mean, median, mode, count) for numerical features. This is crucial to understanding the distribution, variability, and quality of our data.

Data Visualization

```
data['Q10'].value_counts()
```

```
Q10
The city that never sleeps                                    12
The city of love                                              12
city of love                                                  11
Concrete jungle where dreams are made of                       9
City of love                                                   9
                                                              ..
The home of the games.                                         1
Party hard                                                     1
With the sound of waves, Rio de Janeiro sings its song.        1
Jesus statueÂ                                                  1
Secrets travel fast in Paris.                                  1
Name: count, Length: 1231, dtype: int64
```

Figure 3: Visualization of Q10

comments:
`.value_counts()` assists in examining various data points along with count, highlighting the distribution of the dataset.

```
data.boxplot(column='Q8', by='Label')
```

```
<Axes: title={'center': 'Q8'}, xlabel='Label'>
```

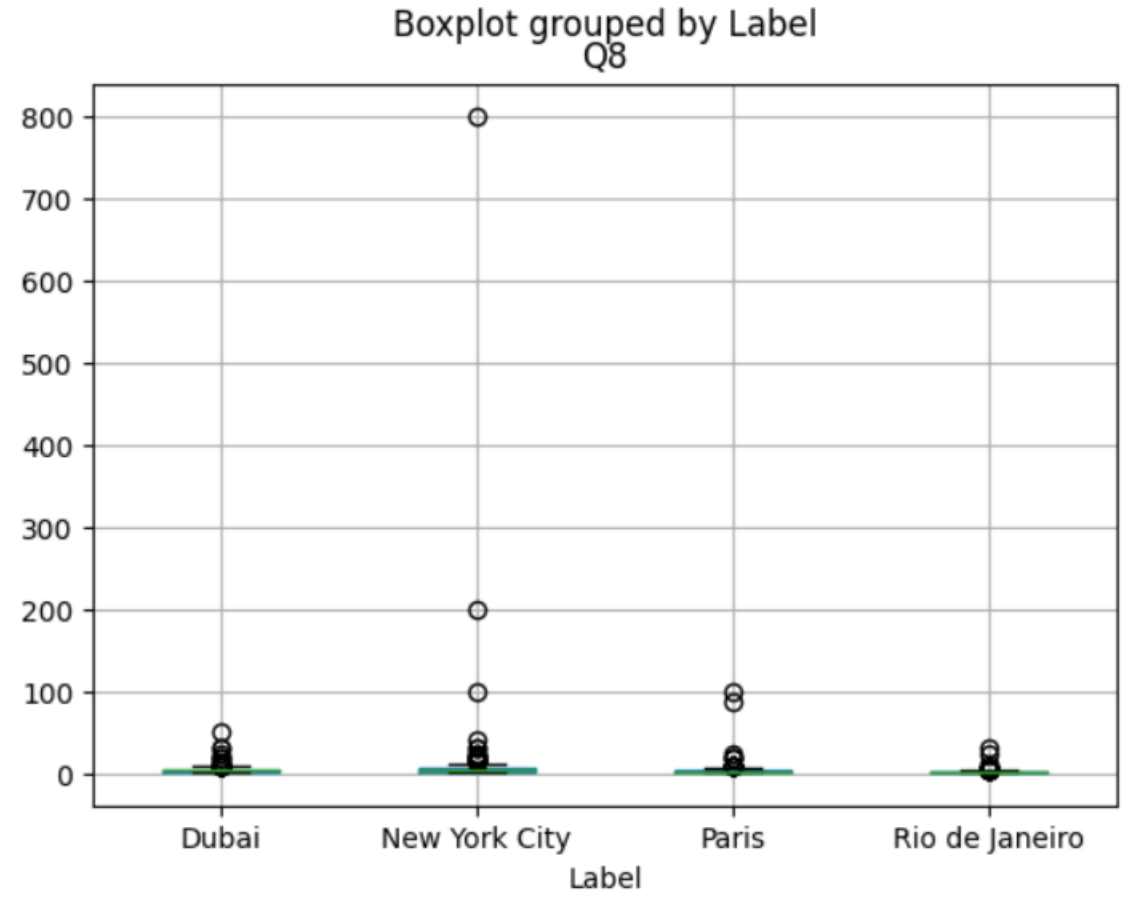

Figure 4: Visualization of Q8.

comments:
The presence of outliers, such as a value of 1000 in the Q8 feature, can lead to several problems in our data analysis and model performance. Outliers can distort measures in the data, like the mean, which can become misleading of the overall representation of the data. Furthermore, this misleading data can affect the outcome of our machine learning model, leading the model to perform poorly because it might overfit to the noise rather than learning the general trend in the data.

We must identify the cause of outliers and determine our imputation strategies.

```
data['Label'].value_counts() #equal number of data points
```

```
Label
Dubai              367
Rio de Janeiro     367
New York City      367
Paris              367
Name: count, dtype: int64
```

Figure 5: Distribution of target variable: cities

comments:
The dataset is balanced with an equal number of data points for each city: Dubai, Rio de Janeiro, New York City, and Paris. Maintaining this balance is crucial to prevent any skewness in the data distribution, which could lead to an imbalanced and biased model. An equitable distribution across city categories ensures that our model remains generalizable and effective in making unbiased predictions across diverse urban contexts

Data Processing

create_pipeline() Function

This function constructs a data pipeline utilizing the `CustomPipeline` class, orchestrating data transformations and incorporating a machine learning model. The pipeline comprises the following stages:

1. **median_imputer**: Imputes missing data in select columns (Q1, Q2, Q3, Q4) using the median of each column.
2. **numeric_converter**: Converts designated columns (Q7, Q8, Q9) to a numeric format, prepping them for computational analysis.
3. **outlier_adjuster**: Modifies outliers within certain columns (Q7, Q8, Q9) by capping values beyond a set threshold, reducing the influence of extreme values.
4. **mean_imputer**: Substitutes missing values in select columns (Q7, Q8, Q9) with the column mean, offering an alternate solution to missing data.
5. **categorical_encoder**: Encodes categorical data from the specified column (Q5) into binary form, making it usable in the modeling process.
6. **rankings_extractor**: Parses ranking details from a specified column (Q6) and formats it for analysis.
7. **text_cleaner**: Standardizes text in the designated column (Q10) by removing unwanted characters and normalizing the case.

8. **CustomTfidfVectorizer**: The CustomTfidfVectorizer class is designed to transform textual data into a numerical format suitable for machine learning models, particularly using the Term Frequency-Inverse Document Frequency (TF-IDF) method. Here's a breakdown of its functionality:

   (a) **Stopwords Removal:** The class begins by reading a list of stopwords from a specified file (`stopwords.txt`). Stopwords are common words like "and," "the," etc., which carry less meaningful information and are often removed before text data processing.

   (b) **Term Frequency Calculation:** Next, the vectorizer calculates the term frequency for each word in the text data of column Q10. Term frequency measures how often a term appears in a document, with each document corresponding to a dataset entry or row.

   (c) **Document Frequency Calculation:** It then computes the document frequency, counting how many documents contain a specific word. This helps gauge the commonality of words across all documents.

   (d) **TF-IDF Calculation:** The TF-IDF value is calculated for each word in each document. TF-IDF increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus (set of all documents). This adjustment accounts for the fact that some words appear more frequently overall. TF-IDF represents words numerically, indicating both their presence and relative importance in a document.

   (e) **Vectorization:** The `CustomTfidfVectorizer` transforms the textual data into a structured, sparse matrix of TF-IDF features. Each row represents a document, while each column corresponds to a word or term from the corpus, with the value being the calculated TF-IDF score.

   This process effectively converts the raw text data into a numerical form making it suitable for training machine learning models like the RandomForest in the pipeline.

Data Representation Features:

- Q1: Number from 1 to 5, where 1 indicates least likely and 5 indicates most likely.
- Q2: Number from 1 to 5, where 1 indicates least likely and 5 indicates most likely.
- Q3: Number from 1 to 5, where 1 indicates least likely and 5 indicates most likely.
- Q4: Number from 1 to 5, where 1 indicates least likely and 5 indicates most likely.
- Q5: Categories of travel companions, represented as binary, split into the following columns:
  - Siblings: Binary (0 = no, 1 = yes)
  - Co-worker: Binary (0 = no, 1 = yes)
  - Partner: Binary (0 = no, 1 = yes)
  - Friends: Binary (0 = no, 1 = yes)

- Q6: Ranking of city aspects, from 1 (least relatable) to 6 (most relatable), split into the following columns:
  - Skyscrapers
  - Sport
  - Art and Music
  - Carnival
  - Cuisine
  - Economic
- Q7: Number representing average temperature.
- Q8: Number of different languages spoken.
- Q9: Number of different fashion styles.
- Q10: Numerical representation of textual data. Split into multiple columns of words (TF-IDF BoW).

Splitting Data into training and testing sets

equal_train_test_split_df Function: The function is designed to partition a dataset into training and testing sets while maintaining an equal distribution of categories, specifically city labels in this case. Here's a detailed description of its operation:

Purpose: The function ensures that each category (city) is represented equally in both the training and testing datasets, preventing any bias that could affect the model's performance and evaluation.

Process:

1. **City List**: It begins by defining a list of cities (Dubai, New York City, Paris, Rio de Janeiro) to ensure these categories are evenly distributed across the split datasets.
2. **Splitting Mechanism**: For each city, the function:
   (a) Filters the dataset to obtain only the records corresponding to the current city.
   (b) Randomly shuffles these records to eliminate any order bias.
   (c) Calculates the number of records to allocate to the test set based on the `test_size` parameter (20% by default).
3. **Index Allocation**: The function then appends the indices of the selected test samples to the `test_indices` list and the remaining to the `train_indices` list.
4. **Dataset Construction**: Utilizing these indices, it constructs the training and testing datasets by locating the respective records in the original dataset.

Outcome: The result is two datasets where each city's presence is proportionally balanced between the training and testing sets. This is crucial for developing a model that is not biased towards a particular category and can generalize well on unseen data.

Model

For the purpose of model exploration, the data was pre-processed in the same way for all explored models. It was altered as follows:

- **Missing Value Handling:** Rows containing missing values in any question were removed to ensure data consistency.
- **Categorical Encoding:** The 'Q5' feature was transformed into binary indicator columns ("Friends," "Coworkers," "Partner," "Siblings") to enable better interpretation by the model. The original feature was subsequently removed.
- **Ranking Extraction:** Rankings within the 'Q6' feature were parsed and extracted into individual columns, providing more granular data for analysis.
- **Data Type Conversion:** Necessary fields were converted to numeric data types to facilitate computation. Train/Test Split: The dataset was divided into a training set (80% of the data) and a testing set (20% of the data), ensuring unbiased model evaluation.

k-Nearest Neighbours

Feature Scaling: To address the sensitivity of k-NN to feature scales, the StandardScaler from scikit-learn was employed. This standardized the features by scaling them to unit variance, ensuring equal contribution to distance calculations.

Parameter Selection: Euclidean distance was selected as the distance metric. Hyperparameter tuning determined that a value of n=6 (number of neighbors) yielded the optimal test accuracy of 0.830.

Logistic Regression

Model and Optimization: The logistic regression model from scikit-learn was fitted to the training data. To account for potential convergence issues, the maximum number of iterations was set to 1000. This provided a reasonable upper limit while potentially allowing for earlier convergence. The resulting test accuracy was 0.895.

Decision Trees

Model and Hyperparameters: A decision tree classifier from scikit-learn was fitted to the training data. Hyperparameter tuning was conducted, focusing on the following:

- `criterion`: min_samples_leaf: Entropy was selected for measuring information gain.
- `min_samples_leaf`: Set to 5 to control tree complexity.
- `splitter`: The "best" strategy was employed to determine the best split at each node.
- Tree depth was left unconstrained for flexibility.

Performance: The decision tree achieved a test accuracy of 0.881.

Random Forest

Model and Hyperparameters: The random forest classifier from scikit-learn was employed for training. Hyperparameter tuning involved the following:

- `max_depth`: To allow for flexibility, no maximum depth was specified.
- `n_estimators`: Set to 100 trees for boosting.
- `min_samples_split`: Set to 2 for splits at internal nodes.
- `Performance`: The optimized random forest model yielded a test accuracy of 0.894.

Performance: The random forest model yielded a test accuracy of 0.894. Logistic regression and random forest classifiers were the most promising so further exploration was done on them, specifically including q10 and fine tuning some parameters.

Modifying Models

To incorporate Question 10 into the predictive models, the following text processing and feature representation approach was adopted:

Text Cleaning: Non-alphanumeric characters were removed from the responses to prepare the data for analysis.

TF-IDF Vectorization: The TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer was employed to numerically represent the words in the text.

Top 100 Features: Only the top 100 most frequent words were converted into features, focusing on the most significant terms.

Stop Words: English stop words (e.g., "the", "but") were removed to eliminate non-informative terms.

Minimum Frequency: A word needed to appear in at least 5 responses to be considered for feature creation.

Updated Random Forest

- **Hyperparameter Tuning**: To optimize the random forest model, a parameter grid was defined, specifying values for the following hyperparameters:
  - `n_estimators`: Number of trees in the forest.
  - `max_depth`: Maximum depth of individual trees.
  - `min_samples_split`: Minimum samples required to split a node.
- **Grid Search**: Grid search was performed to systematically explore the parameter grid, identifying the optimal combination. The best parameters were:
  - No maximum depth (for flexibility)

  - 2 minimum samples to split
  - 50 trees

Performance: This configuration resulted in a test accuracy of 0.91.

Updated Logistic Regression

- Polynomial Features: Polynomial features (degree of 2) were introduced to capture potential non-linear relationships.
- L2 Regularization: L2 regularization was employed to prevent overfitting by penalizing large coefficients. A parameter grid and grid search were used to determine the optimal regularization strength, resulting in a value of 0.01.

Performance: The final logistic regression model achieved a test accuracy of 0.88.

We chose to implement random forest for our custom model. All 6 of the .ipnyb files for the models above will be included in the submission.

Model Choice and Hyperparameters

Model Choice

- **Comparability of Evaluation**: Employing a consistent test set for all evaluated models ensured a fair and unbiased comparison.
- **Evaluation Metrics:** Accuracy was selected as the primary evaluation metric. For balanced classification problems, accuracy offers a clear indication of the proportion of correct predictions made by the model.

Hyperparameter Tuning

Focus: Hyperparameter tuning centered on three key parameters for the random forest model:

- `num_trees`: The number of decision trees within the ensemble.
- `max_depth`: The maximum depth allowed for each individual tree.
- `max_features`: The number of features considered at each split within a tree.

Custom Wrapper: A RandomForestWrapper class was created to streamline the integration of our random forest model with Scikit-learn's GridSearchCV functionality. This enabled efficient exploration of different hyperparameter configurations.

Iterative Refinement: An iterative search strategy was adopted. Starting with broad ranges for each parameter, subsequent iterations progressively narrowed the search space, concentrating on the most promising combinations.

Cross-Validation: To mitigate overfitting and ensure robust evaluation, StratifiedKFold was used for data splitting during grid search. This technique maintains a balanced representation of classes across training and validation folds.

Final Model Configuration

Model and Parameters: The final model, as implemented in pred.py, utilizes a Random Forest classifier with the following hyperparameters:

- `num_trees`: 100
- `max_depth`: 8
- `max_features`: 8

Understanding Our Model: Random Forest

Hyperparameter Significance:

- `num_trees`: A larger number of trees generally leads to better performance, but with diminishing returns and increased computational cost.

- `max_depth`: Deeper trees can capture more complex patterns, but are prone to overfitting.
- `max_features`: Considering a subset of features at each split introduces randomness, reducing overfitting and improving generalization.

Core Component: Decision Trees

- The decision trees form the building blocks of the random forest. They employ entropy calculations to identify optimal splits and determine feature importance for splits.
- The fit method within the DecisionTree class implements the recursive training process.

Predictions, Bootstrapping, and Aggregation

- Predictions initially rely on individual decision trees.
- Bootstrapping (sampling with replacement) and aggregation (averaging/voting) are used to enhance model robustness and reduce variance in predictions.

Workflow and Implementation

- **Pipeline**: A pipeline streamlines the model training process.
- **Data Loading and Random Seed:** Consistency is ensured by reading data and setting a fixed random seed (a key factor during hyperparameter optimization).
- `predict_all` Function: This function encapsulates:
  - Reading test data (specified by filename)
  - Retraining the model on `clean_dataset.csv`
  - Generating predictions and relabeling the test data
  - Returning the modified output

The `predict_all` function is essential in our model's application. It operates on the specified test data (identified by `filename`), reads the data, retrains the model using `clean_dataset.csv`, and then proceeds to predict and relabel the data in the test file, returning the processed output. This function encapsulates the end-to-end prediction process, making it seamless to execute predictions on new datasets.

Prediction

Based on the validation results, we expect our model to achieve an accuracy of 89% on the test set. This point estimate is derived from the consistent performance observed during the cross-validation phase, where the model consistently scored around 88-92% accuracy across different folds.

Empirical Evidence and Reasoning

Result of cross validation.

```
num_trees = 100
Fold accuracy: 0.883495145631068
Fold accuracy: 0.8932038834951457
Fold accuracy: 0.9029126213592233
Fold accuracy: 0.9223300970873787
Fold accuracy: 0.883495145631068
Fold accuracy: 0.8932038834951457
Fold accuracy: 0.8543689320388349
Fold accuracy: 0.8640776699029126
Fold accuracy: 0.9117647058823529
Fold accuracy: 0.8921568627450981
Mean cross-validation accuracy: 0.890100894726823
Test: 0.9136363636363637
```

Figure 6: Cross Validation

After completing cross-validation, we constructed a confusion matrix and conducted an error analysis to identify the sources of prediction errors. This detailed search revealed minor flaws in feature processing and model parameter settings, leading to adjustments that improved the accuracy of the model marginally but notably.

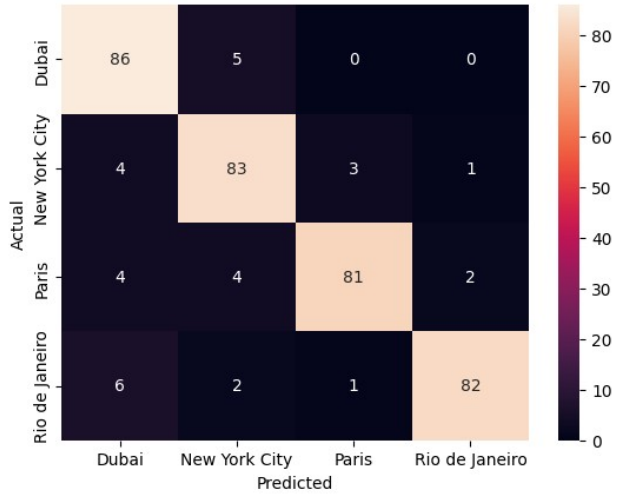

Figure 7: Confusion Matrix

Model Strengths and Limitations

Strengths:

- Random Forest is known for producing high accuracy because it combines the predictions of multiple decision trees, which leads to better perfomance.
- By averaging multiple decision trees, it reduces the risk of overfitting, making it robust against the noise in the data.

Limitations:

- Random Forest models can be quite complex and require more computational resources and memory, especially as the number of trees increases.
- To achieve the best results, random forest requires tuning of various parameters such as number of trees, depth of trees which can be time-consuming.

naseerm4, sohailne,
kanounad, trckovak

Work Distribution

1. Adam: Data Pre-processing, Model Exploration (Random Forest), Model Implementation, Hyper-parameter Fine-tuning, Cross-validation
2. Neha: Data and Model Exploration (Logistic Regression, kNN), **pred.py** script Implementation, Report Writing
3. Katerina: Data Exploration and Visualization
4. Manahil: Model Exploration (Decision Tree), Custom RF Implementation for **pred.py**, Report Writing