Regression and Classification on Restaurant Orders Dataset

Benjamin Ramos
Panther ID: 5793758
Course: CAP4770 Data Mining

February 24, 2025

Abstract

This project explores various regression and classification techniques applied to a restaurant orders dataset. Our objective was to model tipping behavior using different approaches and to classify customer satisfaction and tip categories. Detailed experiments were conducted using PyTorch with both Adam and SGD optimizers, and the report includes key code excerpts, results, and analysis.

Contents

Introduction

This project is divided into five main parts:

1. **Single-Feature Linear Regression:** Predict tip amounts using only the bill amount.
2. **Multiple-Feature Linear Regression:** Improve predictions by incorporating additional features.
3. **Polynomial Regression:** Capture non-linear relationships using polynomial expansions.
4. **Binary Classification:** Predict customer satisfaction (Satisfied vs. Unsatisfied) using logistic regression and a neural network.
5. **Multiple-Classes Classification:** Classify tip amounts into three categories (Low, Medium, High) using One-vs-All (OvA) and One-vs-One (OvO) strategies.

Our implementation in PyTorch included experiments with various hyperparameters and optimizers. Image file names (e.g., `regression_line_adam.png`, `loss_curve_multi_feature.png`, etc.) indicate where figures generated by the scripts should be inserted.

1 Single-Feature Linear Regression

1.1 Objective

Predict the tip amount using a single independent variable. We selected **BillAmount** as the feature and **Tip** as the output, based on the assumption that larger bills might lead to higher tips.

1.2 Implementation

Data Selection & Rationale:

- **Feature:** *BillAmount*
- **Output:** *Tip*

Model Training:
A simple linear regression model was implemented in PyTorch. To explore optimizer behavior, we experimented with both the Adam and SGD (with momentum) optimizers.

```python
# Define the single-feature model using Adam
model_adam = LinearRegressionModel(input_dim=1)
optimizer_adam = optim.Adam(model_adam.parameters(), lr=0.001)
model_adam, loss_adam = train_regression_model(
    model_adam, X_tensor, y_tensor, learning_rate=0.001,
    epochs=1000, print_every=100, weight_decay=0.0,
    early_stopping_patience=None, optimizer=optimizer_adam
)

# Similarly, a model was trained using SGD with momentum:
model_sgd = LinearRegressionModel(input_dim=1)
optimizer_sgd = optim.SGD(model_sgd.parameters(), lr=0.001,
    momentum=0.9)
model_sgd, loss_sgd = train_regression_model(
    model_sgd, X_tensor, y_tensor, learning_rate=0.001,
    epochs=1000, print_every=100, weight_decay=0.0,
    early_stopping_patience=None, optimizer=optimizer_sgd
)
```

Visualization:
A scatter plot of the data with the regression line (using Adam) was produced.

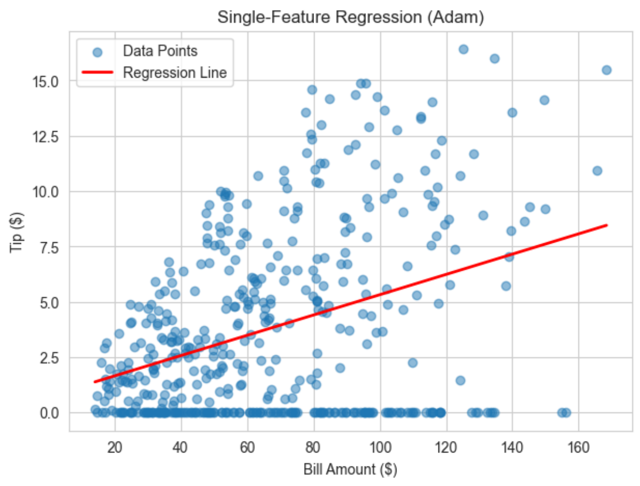


Figure 1: Regression Line using Adam Optimizer

1.3 Results & Analysis

- **Final Normalized MSE:** Approximately 0.8613 for Adam and 0.8608 for SGD.
- **Unnormalized $\mathbf{R^2}$:** Approximately 0.14.

Observation:

Our experiments with single-feature regression, using BillAmount to predict Tip, revealed that while the model converged steadily with both Adam and SGD optimizers, the overall performance remained limited. Both optimizers achieved a similar normalized Mean Squared Error (MSE) of approximately 0.86, resulting in an unnormalized R^2 value of only around 0.14. This low R^2 suggests that BillAmount alone accounts for roughly 14% of the variability in tip amounts.

The comparable performance between Adam and SGD—with SGD even converging rapidly—highlights that, at this scale, optimizer choice did not drastically influence outcomes. However, the inherent limitation of a single feature indicates that additional predictors are essential to capture the complexity of tipping behavior.

2 Multiple-Feature Linear Regression

2.1 Objective

Enhance prediction accuracy by using three features: **BillAmount**, **NumItems**, and **Tip_Percentage**, with **Tip** as the output.

2.2 Implementation

Data Selection & Rationale:

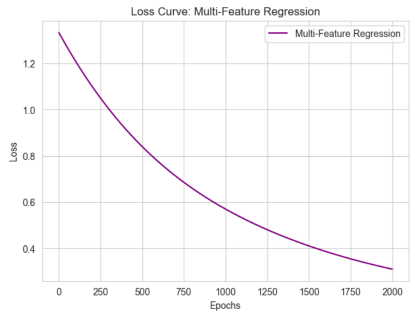
- **BillAmount:** Directly related to tip magnitude.
- **NumItems:** Indicates order complexity.
- **Tip_Percentage:** Captures the relative generosity of the tip.

Model Training:
The multi-feature linear regression model was built and trained after normalizing the data and splitting it into 80% training and 20% validation sets.

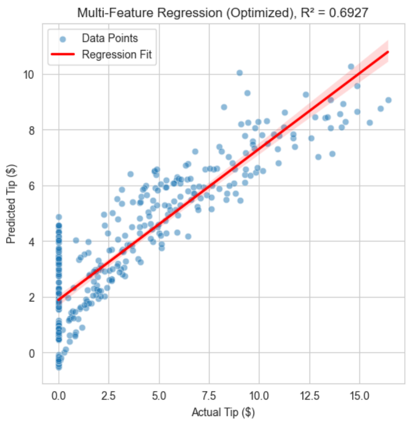
```python
features = ["BillAmount", "NumItems", "Tip_Percentage"]
target_col = "Tip"
X_tensor, y_tensor, X_mean, X_std, y_mean, y_std = preprocess_data(
    df, features, target_col, normalize=True)

model_multi = LinearRegressionModel(input_dim=len(features))
X_train, X_val, y_train, y_val = train_test_split(X_tensor,
    y_tensor, test_size=0.2, random_state=0)
model_multi, loss_multi = train_regression_model(
    model_multi, X_train, y_train, learning_rate=0.0003,
    epochs=2000, print_every=200, weight_decay=0.001
)
```

Visualizations:
The loss curve and actual vs. predicted tips are combined into a single figure on one page.



(a) Loss Curve for Multi-Feature Regression



(b) Actual vs. Predicted Tips

Figure 2: Multi-Feature Regression Visualizations

2.3 Results & Analysis

- **Training MSE (normalized):** Approximately 0.3099.
- **Validation MSE (normalized):** Approximately 0.2340.
- **Unnormalized $\mathbf{R^2}$:** Approximately 0.69.
- **Learned Parameters:** Weights $\approx \begin{bmatrix} 0.10978 & 0.30074 & 0.48728 \end{bmatrix}$, Bias ≈ 0.05525.

Incorporating multiple features—specifically BillAmount, NumItems, and Ti_Percentage—resulted in a significant improvement in predictive performance. The multi-feature model achieved a normalized MSE of approximately 0.31 on the training set and 0.23 on the validation set, with an unnormalized R^2 of about 0.69. This indicates that nearly 70% of the variance in tip amounts can be explained when these three features are considered together.

The learned weights (approximately [0.11, 0.30, 0.49]) provide insights into the relative contribution of each feature. For instance, a higher weight on Tip_Percentage suggests that the proportion of the bill given as a tip is a strong predictor. Overall, this model demonstrates that a more comprehensive feature set can substantially enhance model accuracy compared to using a single predictor.

3 Polynomial Regression

3.1 Objective

Investigate non-linear relationships by applying polynomial regression to the three features from Part 2. Models were built for polynomial degrees 2, 4, and 6.

3.2 Implementation

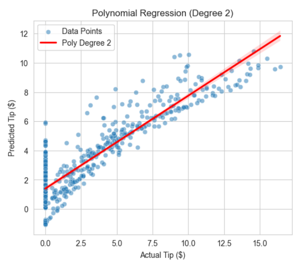
Feature Transformation:
Polynomial features (without cross-terms) were generated by raising each feature to powers from 1 to the desired degree.

```python
def polynomial_features(X, degree):
    poly_terms = [X ** d for d in range(1, degree + 1)]
    return torch.cat(poly_terms, dim=1)
```

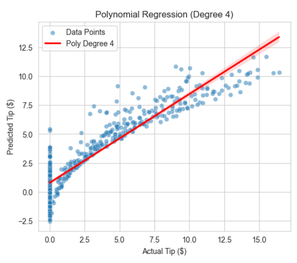
Model Training:
Separate models were trained for each polynomial degree. For the degree 6 model, the learning rate and weight decay were adjusted to counteract instability.

```python
for degree in degrees:
    print(f"\n===␣Polynomial␣Degree␣{degree}␣===")
    X_poly = polynomial_features(X_tensor, degree)
    lr_poly = 0.001 if degree < 6 else 0.0005
    wd_poly = 0.01 if degree < 6 else 0.05
    model_poly = LinearRegressionModel(input_dim=X_poly.shape[1])
    model_poly, loss_poly = train_regression_model(
        model_poly, X_poly, y_tensor, learning_rate=lr_poly, epochs
            =1000,
        print_every=200, weight_decay=wd_poly,
            early_stopping_patience=200
    )
```

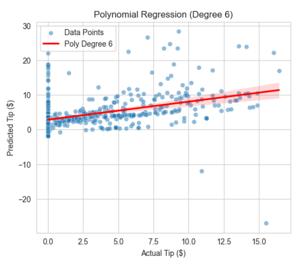
Visualizations:
The three polynomial regression plots are grouped together on one page.



(a) Degree 2



(b) Degree 4



(c) Degree 6

Figure 3: Polynomial Regression Visualizations

3.3 Analysis

Polynomial regression was used to explore potential non-linear relationships by expanding the feature set to higher polynomial degrees. Our experiments with degrees 2, 4, and 6 revealed an interesting trade-off:

- Degree 2 Model: The degree 2 model improved the unnormalized R^2 to approximately 0.76, indicating that introducing a quadratic term captured additional variance in the data compared to the linear model.
- Degree 4 Model: Further increasing the degree to 4 resulted in an even better fit (R^2 0.83). This model was able to capture more nuanced non-linear patterns in the relationship between the predictors and tip amount.
- Degree 6 Model: The degree 6 model, however, performed very poorly, with a normalized MSE rising to about 1.56 and an unnormalized R^2 of -0.57. A negative R^2 means that the model's predictions are worse than simply predicting the mean tip. This dramatic degradation is a clear sign of overfitting: the model becomes too flexible, capturing noise rather than the underlying trend.

These results underscore the importance of balancing model complexity with generalization. While moderate-degree polynomials (degree 2 and 4) enhance the model's capacity to capture non-linear relationships, pushing the degree too high introduces instability and poor performance.

4 Binary Classification

4.1 Objective

Predict customer satisfaction by constructing a binary label from the dataset.

4.2 Implementation

Label Construction:

```python
df["BinarySatisfaction"] = df["CustomerSatisfaction"].apply(lambda
    x: 1 if x == "Satisfied" else 0)
```

Data Splitting & Preprocessing:
The data was split into 80% training and 20% testing sets with normalized features.

Model Training:
Two models were built:

- Logistic Regression
- Neural Network (MLP)

For example, training the logistic regression model:

```python
model_logistic = LogisticRegressionClassifier(input_dim=X_train.
    shape[1])
model_logistic = train_classification_model(model_logistic, X_train
    , y_train,
    learning_rate=0.001, epochs=500, print_every=50)
```

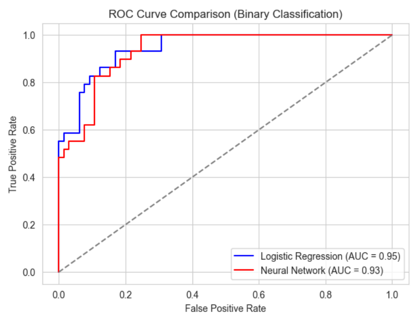
Results:
Logistic Regression:

- Accuracy: Approximately 77.7%
- Precision: Approximately 58%
- Recall: 100%
- F1-Score: Approximately 73%

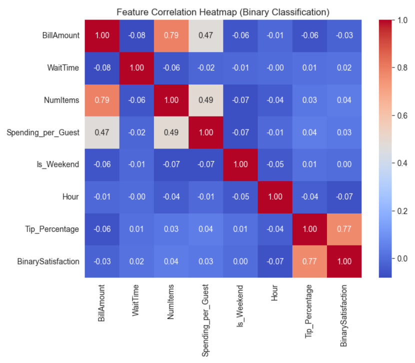
MLP Classifier:

- Accuracy: Approximately 84.0%
- Precision: Approximately 68%
- Recall: Approximately 90%
- F1-Score: Approximately 78%

Visualizations:



(a) ROC Curve Comparison



(b) Correlation Heatmap

Figure 4: Binary Classification Visualizations: ROC Curve and Correlation Heatmap

4.3 Analysis

For binary classification, the objective was to predict customer satisfaction (Satisfied vs. Unsatisfied) based on various features. Two models were implemented: a simple logistic regression model and a neural network (MLP) with one hidden layer.

- Logistic Regression: This model achieved an accuracy of approximately 77.7%, with a perfect recall (100%) but a moderate precision (58%). The high recall indicates that the model is very good at detecting positive cases (i.e., it rarely misses a satisfied customer), but the lower precision suggests it also produces a fair number of false positives.

- Neural Network (MLP): The MLP outperformed the logistic regression, reaching an accuracy of about 84% and improving precision (around 68%) while maintaining high recall (approximately 90%). The non-linear transformations introduced by the hidden layer allow the MLP to capture complex relationships that a simple linear model might miss.

Visualizations such as ROC curves and confusion matrices provided further insight into the models' performance. The ROC curves demonstrated good discrimination ability, and the confusion matrix highlighted areas for potential improvement—such as adjusting the decision threshold to balance precision and recall more effectively.

5 Multiple-Classes Classification

5.1 Objective

Derive a multi-class label by categorizing tip amounts into three groups and implement classification using both One-vs-All (OvA) and One-vs-One (OvO) logistic regression strategies.

5.2 Implementation

Label Derivation:
Tip amounts were categorized as follows:

- **Low Tip (0):** Tips < \$2.00
- **Medium Tip (1):** Tips between \$2.00 and \$5.00
- **High Tip (2):** Tips > \$5.00

```
def categorize_tip(tip_value):
    if tip_value < 2.0:
        return 0
    elif tip_value <= 5.0:
        return 1
    else:
        return 2
df["TipCategory"] = df["Tip"].apply(categorize_tip)
```

Strategies:

- **One-vs-All (OvA):** Train one binary classifier per class.
- **One-vs-One (OvO):** Train binary classifiers for each pair of classes and use majority voting for the final prediction.

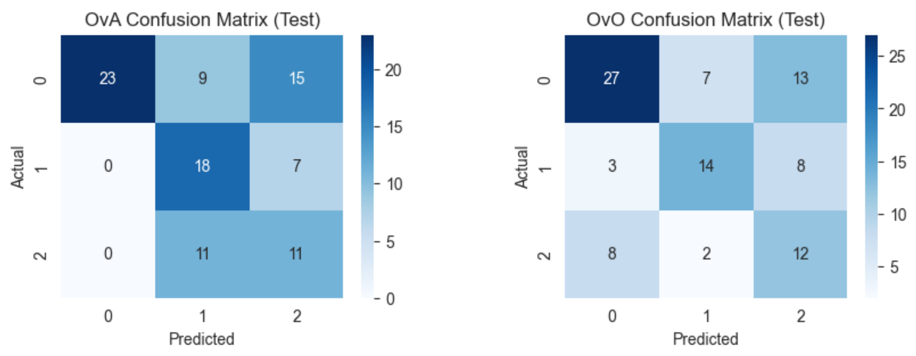
```
# OvA strategy:
ova_models = train_ova_classifiers(X_train_multi, y_train_multi,
    num_classes, lr=0.001, epochs=500)
ova_preds_test = predict_ova(ova_models, X_test_multi)
```

```
# OvO strategy:
ovo_models = train_ovo_classifiers(X_train_multi, y_train_multi,
    num_classes, lr=0.001, epochs=500)
ovo_preds_test = predict_ovo(ovo_models, X_test_multi, num_classes)
```

Results:

- **OvA Accuracy:** Approximately 55.32%
- **OvO Accuracy:** Approximately 56.38%

Visualizations:



(a) Confusion Matrix — OvA (b) Confusion Matrix — OvO

Figure 5: Confusion Matrices for Multi-Classes Classification

5.3 Analysis

The multi-class classification task involved categorizing tip amounts into three groups: Low, Medium, and High. Two strategies were explored:

- One-vs-All (OvA): In this approach, a separate binary classifier was trained for each class. The final class prediction was made by selecting the classifier with the highest output score. The OvA approach yielded an accuracy of approximately 55.32%. This method is relatively straightforward and scales linearly with the number of classes.

- One-vs-One (OvO): The OvO strategy involved training a classifier for each pair of classes and then using a majority voting scheme to determine the final prediction. This approach achieved a slightly higher accuracy of around 56.38%. Although OvO can potentially capture finer distinctions between class pairs, it increases the number of classifiers substantially as the number of classes grows.

Comparative Insights: While both OvA and OvO achieved similar performance levels (around 55–56% accuracy), their complexity differs significantly. The modest accuracy observed for both strategies suggests that the features currently available may not be sufficiently discriminative for differentiating between the tipping categories. Future improvements could include additional feature engineering or adopting a direct multi-class classifier with a softmax layer, which may provide a more robust solution for this task.

6 Conclusion

Overall, this project provided a thorough exploration of regression and classification methods applied to the restaurant orders dataset. Several key insights emerged:

Regression Insights:

- The single-feature linear regression using BillAmount as the sole predictor demonstrated that while there is a relationship between the bill and tip, the model could only explain around 14% of the variability in tips.
- By incorporating additional features—NumItems and Tip_Percentage—the multi-feature linear regression model significantly improved performance, explaining nearly 70% of the tip variance. This highlights the critical role of feature selection in enhancing model accuracy.
- Polynomial regression further revealed that moderate complexity (degree 2 and degree 4) can capture non-linear relationships effectively, increasing the R^2 to 0.76 and 0.83, respectively. However, pushing the model to a higher degree (degree 6) led to severe overfitting, as demonstrated by a negative R^2, underscoring the importance of balancing flexibility and generalization.

Classification Insights:

- In binary classification, the implementation of both logistic regression and an MLP showed that non-linear models can better capture the underlying patterns in customer satisfaction. The MLP achieved higher accuracy and better overall performance metrics, suggesting that the relationships in the data are more complex than what a linear model can capture.
- For multi-class classification, both the One-vs-All and One-vs-One strategies produced similar, modest accuracies (around 55–56%). This outcome indicates that the current feature set may not be sufficiently discriminative to effectively separate the derived tipping categories. More sophisticated approaches or additional features may be needed to improve performance in multi-class settings.

Regression Insights:

- The experiments underscore the importance of using a diverse set of features and carefully tuning model complexity and regularization to avoid overfitting.
- Future work could explore additional predictors (such as time-based variables, customer demographics, or contextual data) and experiment with more advanced architectures (e.g., deep neural networks or ensemble methods) to further improve predictive performance.

- For multi-class classification, moving toward a direct softmax-based model might simplify the approach and potentially yield better results compared to decomposing the problem into multiple binary tasks.

In summary, the project not only met the established objectives but also provided valuable insights into model behavior, the trade-offs involved in increasing complexity, and the challenges of classification with limited features. These findings lay a solid foundation for further refinement and exploration in predictive modeling within the restaurant domain.