ASUENG

Artificial Intelligence

Presented by :

Ahmed Mohamed Elsayed Tabbash	20P1076
Ahmed Ibrahim Eldera	20p2701
Mohamed Awad Shawki	22P0240
Paula Emil Alexan	2200750
YAMAN ABDELHAMID ABDELSAMIE ELEWA	2002078

Presented for :

Dr. Maryam Nabil

Eng. Mohammed Essam

Junior - Spring 2025 - Artificial intelligence - CSE472

Contents

1. Data Cleaning & Preprocessing (`clean.py` and tryout.ipynb)

Why Clean and Preprocess Data?

Machine learning models require:

- **Numerical input** (no text or categorical data).
- **No missing values** (NaNs).
- **Consistent feature formats** (e.g., all sizes in MB).
- **No outliers or erroneous data** (e.g., typos, impossible values).

Step-by-Step Breakdown

a. Column Renaming

- **Purpose:** Makes the data human-readable and easier to work with.
- **Example:** `X0` → `AppName`, `X1` → `Category`, etc.

b. Dropping Unnecessary Columns

- **Purpose:** Removes columns that do not help prediction (e.g., `AppName` is just an identifier).

c. Category Handling

- **Remove Erroneous Categories:** E.g., a category labeled `'1.9'` is likely a data entry error.
- **Group Rare Categories:** Categories with very few samples can cause overfitting. Grouping them into `'OTHER'` ensures the model doesn't learn noise.
- **One-Hot Encoding:** Converts each category into a binary column (e.g., `Cat_BUSINESS` = 1 if the app is business, else 0). This allows models to use categorical data.

d. Numeric Conversion

- **NumReviews, AppSize, NumInstalls, Price:** All must be numeric.
- **AppSize:** Converts all sizes to MB (e.g., `12k` → `0.0117 MB`, `20M` → `20 MB`). Handles missing or ambiguous values by filling with the median.
- **NumInstalls:** Removes `+` and commas, converts to integer.
- **Price:** Ensures all prices are numeric.

e. Boolean and Categorical Encoding

- **IsFree:** Converts `"Free"` to 0 and `"Paid"` to 1.
- **AgeCategory:** One-hot encodes age restrictions (e.g., `Age_Everyone`, `Age_Teen`).
- **Genres:** Apps can have multiple genres (e.g., `"Action;Adventure"`). Uses `MultiLabelBinarizer` to create a column for each genre, set to 1 if the app has that genre.

f. Date Handling

- **LastUpdate:** Converts to datetime, extracts the year (e.g., `2018`), then drops the original column.

g. Dropping More Columns

- **Version, MinAndroidVer:** Often too granular or inconsistent for modeling, so they are dropped.

h. Handling Missing Values

- **AppSize:** Fill with median.
- **Rating (target):** Drop rows with missing ratings (since you can't train on them).
- **Other features:** Ensure no missing values remain.

i. Normalization

- **NumInstalls, NumReviews:** These are often highly skewed (some apps have millions of installs, most have few). Applying `np.log1p()` (logarithm of 1 + value) compresses large values and spreads out small ones, making the data easier for models to learn from.

j. Final Checks

- **No NaNs:** Ensures all missing values are handled.
- **No text columns:** All features must be numeric for ML models.
- **No infinite values:** Ensures no division-by-zero or log(0) errors.

k. Saving

- **Cleaned Data:** Saved for use in training and testing.

---

2. Model Training & Evaluation (`Train.py`)

Why Train Multiple Models?

- **No single model is best for all problems.**
- **Trying a variety of models** (linear, tree-based, instance-based, etc.) helps find the best fit for your data.

Step-by-Step Breakdown

a. Data Loading

- Loads the cleaned, normalized data for training, validation, and testing.

b. Feature Selection

- Selects only the columns used for prediction (excludes the target `Rating` and any identifiers).

c. Target Extraction

- Sets the `Rating` column as the value to predict.

d. Missing Value Checks

- Ensures no missing values in features or targets.

e. LazyML (LazyPredict)

- **What is it?** A library that quickly trains and evaluates many regression models with default settings.
- **Why use it?** To get a fast, broad comparison of many algorithms and see which ones are promising.
- **How does it work?** It fits each model on the training data and evaluates on the validation set, reporting metrics like R^2 and RMSE.

f. K-Fold Cross-Validation

- **What is it?** A robust way to estimate model performance.
- **How does it work?**
  - Splits the training data into `k` (e.g., 5) folds.
  - Trains the model on `k-1` folds, tests on the remaining fold.
  - Repeats this `k` times, each time with a different test fold.
  - Reports the average performance.
- **Why use it?** Reduces the risk of overfitting to a particular train/validation split and gives a more reliable estimate of model performance.

g. Model Training & Evaluation

- **Trains each model on the full training set.**
- **Evaluates on validation and test sets using Mean Squared Error (MSE).**
- **Saves each trained model for later use.**

Models Used:

1. LinearRegression

- **How it works:** Finds the best-fitting straight line (hyperplane) through the data.
- **When to use:** When you suspect a linear relationship between features and target.
- **Pros:** Simple, interpretable.
- **Cons:** Can't capture non-linear relationships.

2. Ridge Regression

- **How it works:** Like LinearRegression, but adds L2 regularization (penalizes large coefficients).
- **Why:** Helps prevent overfitting, especially when features are correlated.

3. Lasso Regression

- **How it works:** Like Ridge, but uses L1 regularization (can shrink some coefficients to zero, effectively selecting features).
- **Why:** Useful for feature selection and preventing overfitting.

4. RandomForestRegressor

- **How it works:** Builds many decision trees on random subsets of the data and averages their predictions.
- **Why:** Handles non-linearities, interactions, and is robust to outliers and overfitting.

5. GradientBoostingRegressor

- **How it works:** Builds trees sequentially, each one correcting the errors of the previous.
- **Why:** Often achieves high accuracy, especially on tabular data.

6. KNeighborsRegressor

- **How it works:** Predicts the target by averaging the values of the k nearest neighbors in feature space.
- **Why:** Simple, non-parametric, can capture local patterns.

7. SVR (Support Vector Regression)

- **How it works:** Tries to fit as many data points as possible within a margin, using kernel tricks to capture non-linear relationships.
- **Why:** Good for complex, non-linear data, robust to outliers.

Evaluation Metrics

Mean Squared Error (MSE)

- **Definition:** The average of the squared differences between predicted and actual values.
- **Why:** Penalizes large errors more than small ones, commonly used for regression.

Cross-Validation MSE

- **Definition:** The average MSE across all folds in K-Fold CV.
- **Why:** Gives a more robust estimate of model performance.

---

3. Prediction (`prediction.py`)

Purpose

- **Apply a trained model to new, unseen data** (e.g., for a competition or real-world deployment).

Step-by-Step Breakdown

a. Load Test Data

- Reads the cleaned and normalized test data.

b. Feature Alignment

- Ensures the test data has the same features as the training data.
- Handles missing columns by filling with zeros (so the model can still make predictions).

c. Load Model

- Loads a previously trained model (e.g., Lasso, RandomForest) using `joblib`.

d. Predict

- Uses the model to predict ratings for the test data.

e. Save Results

- Outputs predictions to a CSV file for submission or further analysis.

---

4. Interactive Data Cleaning (`tryout.ipynb`)

Purpose

- **Prototype and visualize** each cleaning step.
- **Debug and explore** the data interactively.
- **Document** the cleaning process.

Why Use a Notebook?

- You can see the effect of each transformation.
- Easy to plot, summarize, and check data at each step.
- Useful for developing and testing your cleaning pipeline before scripting it in clean.py.

How Everything Fits Together

1. **Raw Data** → `clean.py`/`tryout.ipynb` → **Cleaned Data**
2. **Cleaned Data** → `Train.py` → **Trained Models**
3. **Trained Models + New Data** → `prediction.py` → **Predictions**

Why This Pipeline?

- **Data cleaning** ensures the models get the best possible input.
- **Trying multiple models** increases the chance of finding the best fit for your data.
- **Cross-validation** ensures your results are robust and not due to chance.
- **Saving models** allows for easy deployment and reuse.
- **Automated prediction** enables you to apply your solution to new data quickly.

Summary Table

File	Purpose	Key Steps/Models Used
clean.py	Data cleaning & preprocessing	Renaming, encoding, normalization, missing value handling
Train.py	Model training, validation, evaluation	Linear, Ridge, Lasso, RandomForest, GradientBoosting, KNN, SVR, LazyML, K-Fold CV
prediction.py	Predicting on new/test data	Loads model, aligns features, predicts, saves results
tryout.ipynb	Interactive data cleaning & exploration	Step-by-step cleaning, encoding, normalization, splitting, saving