**Machine Learning – Assessment 2 ‘Individual Assignment Submission’**

**Samuel Vierny 30/11/2024**

**Short Analysis of the Competitions**

**Project 1:** [**Salary Prediction for Job Postings**](https://www.kaggle.com/competitions/salary-prediction-for-job-postings)

* **Type of ML Task:** Regression
  + Predicting numerical values (salaries) based on job posting data.
* **Dataset:**
  + Contains job posting metadata like job descriptions, titles, and possibly categorical features like location or industry.
  + Preprocessing involves handling text data (NLP methods), encoding categorical variables, and managing missing values.
* **Benefits:**
  + Straightforward regression task for beginners, focusing on common preprocessing techniques like handling text and categorical data.
  + Practical real-world application, making results interpretable and engaging.
  + Lower computational overhead compared to tasks with unstructured or large datasets (e.g., images or audio).
* **Limitations:**
  + Text data requires careful preprocessing, which might introduce complexity (e.g., tokenization, embeddings).
  + Possible skewed salary distribution could affect model performance.

**Project 2:** [**Music Hackathon**](https://www.kaggle.com/competitions/MusicHackathon)

* **Type of ML Task:** Regression or Classification
  + Could involve predicting numerical music properties or classifying genres or user preferences.
* **Dataset:**
  + Includes audio features (e.g., tempo, loudness) and metadata.
  + Requires preprocessing of audio data using techniques like spectral analysis or feature extraction (e.g., MFCCs).
* **Benefits:**
  + Engaging and creative domain, appealing for music enthusiasts.
  + Opportunity to work with advanced techniques like deep learning for audio data.
* **Limitations:**
  + High computational requirements, especially for audio file preprocessing.
  + Audio data preprocessing requires specialized knowledge of signal processing.

**Project 3:** [**Classifying the Brain on Music**](https://www.kaggle.com/competitions/classifying-the-brain-on-music)

* **Type of ML Task:** Classification
  + Predict brain activity patterns based on music-related stimuli.
* **Dataset:**
  + Likely includes EEG or fMRI data, which are high-dimensional time-series signals.
  + Preprocessing involves noise filtering, feature extraction, and dimensionality reduction.
* **Benefits:**
  + Unique interdisciplinary challenge combining neuroscience and machine learning.
  + Opportunity to apply techniques for time-series or high-dimensional data.
* **Limitations:**
  + Very complex for a first project, requiring expertise in both neuroscience and ML.
  + Computationally expensive and prone to overfitting due to the high-dimensional nature of the data.

**Why Project 1 is the Best Fit for You**

* **Alignment with Experience Level:**
  + Project 1 has manageable complexity for a first task, with data that’s easy to understand and preprocess.
* **Focus on Core ML Skills:**
  + It allows you to focus on supervised learning fundamentals, such as feature engineering, model tuning, and evaluation, without being overwhelmed by domain-specific complexities.
* **Balanced Challenge:**
  + Preprocessing job descriptions (text data) introduces a challenge without being overly complex, making it a good learning opportunity.
* **Interpretability of Results:**
  + Predicting salaries is a relatable and interpretable task, which helps in understanding model performance and potential biases.
* **Computational Requirements:**
  + The dataset is structured and less computationally demanding compared to audio or neuroscience datasets, making it feasible for faster experimentation.
* **Opportunity for Feature Engineering:**
  + The inclusion of job descriptions and skills presents an opportunity to explore natural language processing techniques, such as tokenization or the use of embeddings. This enhances your ability to work with unstructured data.
  + Dealing with High-Cardinality Categorical Variables: Features like 'Company' and 'Job Title' have many unique values, allowing you to experiment with encoding methods like frequency encoding or target encoding. This helps in understanding how to handle variables that could otherwise introduce sparsity into the dataset.
* Passion subject:
  + While Music related assessments might be more related to my passions and interests, Project 1 still realistically the most relevant and fitting for my assessment. I will endeavour to take on Projects 2 and 3 another time or in the future once I have strengthened my machine learning skills.

**Project 1 start:**

**Overview: Step-by-Step Plan for Project 1**

Below is a written overview of the steps we’ll take for **Project 1: Salary Prediction for Job Postings**. Each step builds on the fundamentals of machine learning, ensuring clarity and organization.

**Step 1: Data Understanding and Exploration**

* **Goal:** Load the dataset, understand its structure, and explore the key features.
* **Reason:** It’s critical to know what type of data you’re working with (e.g., numerical, categorical, text) before preprocessing and modeling.

**Step 2: Data Cleaning**

* **Goal:** Handle missing values, duplicates, and outliers in the dataset.
* **Reason:** Clean data ensures more reliable and accurate model performance.

**Step 3: Feature Engineering**

* **Goal:** Transform text data into usable numerical formats (e.g., TF-IDF), encode categorical variables, and scale numerical features if needed.
* **Reason:** Machine learning models require numerical input, and effective feature engineering improves prediction accuracy.

**Step 4: Splitting Data**

* **Goal:** Split the dataset into training and testing sets.
* **Reason:** Evaluate the model on unseen data to check for overfitting and ensure generalization.

**Step 5: Baseline Model**

* **Goal:** Train a simple regression model (e.g., Linear Regression) as a baseline.
* **Reason:** Establish a starting point for model performance to compare with advanced models.

**Report: Exploratory Data Analysis and Data Cleaning**

**Approach to Developing the Model**

1. **Exploratory Data Analysis (EDA):**
   * The dataset was loaded and examined for its structure, types of features, and potential issues.
   * Summary statistics were calculated for numerical columns to understand their distributions, ranges, and outliers.
   * A histogram was created to visualize the distribution of the target variable, Mean\_Salary. This provided insights into its skewness, variability, and range.
2. **Data Cleaning:**
   * Missing values were identified across both numerical and categorical columns. The missing value analysis highlighted columns with significant gaps, such as Profile and Remote, as well as others with more moderate amounts of missing data.
   * For **numerical columns**, missing values were imputed with the median. This approach was chosen to reduce the influence of outliers that could distort the mean.
   * For **categorical columns**, missing values were replaced with the placeholder "unknown". This ensured the missing values were explicitly recognized and encoded as part of the dataset rather than being ignored or arbitrarily filled.
3. **Feature Engineering and Preprocessing:**
   * Categorical columns were converted into numerical representations using OneHotEncoder, ensuring that the machine learning models could interpret the data effectively.
   * Numerical columns were standardized using StandardScaler to bring all features to a similar scale. This step was crucial for models sensitive to feature magnitudes, such as linear regression or support vector machines.
   * The processed numerical and categorical data were combined into a unified feature matrix for downstream modeling.
4. **Train-Test Split:**
   * The cleaned and preprocessed dataset was split into training and validation sets (80%-20%). This split ensured the model could be evaluated on unseen data to assess its generalization performance.

**Reflections on the Process**

1. **Exploratory Data Analysis (EDA):**
   * The EDA phase provided critical insights into the dataset. For example, the Mean\_Salary target variable showed a wide range of values, highlighting potential outliers and skewness that could impact model performance.
   * Identifying columns with missing values early in the process helped establish a clear data-cleaning strategy.
2. **Handling Missing Values:**
   * Imputing missing values for numerical columns with the median was straightforward and effective, particularly in mitigating the effects of outliers.
   * Replacing missing categorical values with "unknown" ensured the dataset remained complete while preserving the possibility for the model to treat these cases as meaningful.
3. **Feature Engineering:**
   * Using OneHotEncoder for categorical variables ensured no category was left out or incorrectly prioritized. However, it also increased the dimensionality of the dataset, which might lead to computational challenges in more complex models.
   * Standardizing numerical features was a necessary step for consistency but added some preprocessing complexity.
4. **Modeling Considerations:**
   * The structured and preprocessed dataset provided a strong foundation for building supervised machine learning models.
   * The train-test split ensured the model’s performance could be evaluated fairly, but further steps such as cross-validation might be considered for more robust evaluation.

This stage of the project focused on ensuring the data was clean, complete, and in a format suitable for modeling. The choices made in handling missing data, encoding categorical features, and scaling numerical data reflect best practices in data preprocessing. These steps established a robust pipeline for downstream machine learning tasks.

**Why PyCaret?**

1. **Ease of Use:**
   * PyCaret automates the workflow for comparing multiple models, making it much faster to identify the best-performing algorithms.
2. **Built-in Comparison:**
   * With a single command, PyCaret ranks models based on performance metrics like MAE, RMSE, or R².
3. **Hyperparameter Tuning:**
   * PyCaret supports automated hyperparameter tuning for selected models, simplifying optimization.
4. **Model Interpretability:**
   * PyCaret includes tools to analyze model behavior, like feature importance and residual plots.

**Problem 1: Breakdown:**

1. **High Dimensionality:** One-hot encoding has created many features (as seen in the shape (98806, 26598)), significantly increasing the size of the dataset.
2. **Memory Constraints:** Your machine doesn't have sufficient memory to handle this large array for PyCaret operations.

**Solutions to Address the Issue:**

**1. Reduce One-Hot Encoded Features:**

* Apply dimensionality reduction to the categorical features by limiting the number of unique categories.
* Use **target encoding** or **frequency encoding** instead of one-hot encoding for high-cardinality features.

**1. Organize Feature Importance Results**

**Top Features Based on Random Forest Importance**

| **Rank** | **Feature** | **Importance Score** |
| --- | --- | --- |
| 1 | **Jobs\_Group** | 0.227599 |
| 2 | **Profile** | 0.089237 |
| 3 | **State** | 0.064870 |
| 4 | **Skills** | 0.064450 |
| 5 | Company | 0.059001 |
| 6 | City | 0.056630 |
| 7 | Job | 0.054288 |
| 8 | Reviews | 0.049716 |
| 9 | Location | 0.048628 |
| 10 | Director\_Score | 0.043392 |
| 11 | ID | 0.037471 |
| 12 | Sector | 0.036041 |
| 13 | Company\_Score | 0.035093 |
| 14 | Sector\_Group | 0.030464 |
| 15 | Frecuency\_Salary | 0.027210 |
| 16 | URL | 0.019253 |
| 17 | Director | 0.017382 |
| 18 | Revenue | 0.014908 |
| 19 | Employee | 0.012826 |
| 20 | Remote | 0.011542 |

**Top Features Based on Recursive Feature Elimination (RFE)**

| **Rank** | **Feature** |
| --- | --- |
| 1 | **Frecuency\_Salary** |
| 1 | **Jobs\_Group** |
| 1 | **Profile** |
| 1 | **Remote** |
| 1 | **Company\_Score** |
| 1 | **State** |
| 1 | **Director\_Score** |
| 1 | **Sector\_Group** |
| 1 | **Revenue** |
| 1 | **Employee** |
| 2 | Sector |
| 3 | City |
| 4 | Skills |
| 5 | Director |
| 6 | Company |
| 7 | URL |
| 8 | Reviews |
| 9 | Location |
| 10 | Job |
| 11 | ID |

**Note**: In RFE, a rank of 1 indicates the most important features.

**Overlap Between Random Forest and RFE Top Features**

* **Jobs\_Group**
* **Profile**
* **State**
* **Skills**

These features appear in the top rankings of both methods, suggesting they are critical to retain.

**3. Summarize Unique Categories in Categorical Columns**

| **Feature** | **Unique Categories** |
| --- | --- |
| ID | 33,248 |
| Job | 17,227 |
| Company | 13,996 |
| Location | 12,543 |
| Skills | 10,805 |
| URL | 5,148 |
| City | 2,952 |
| Director | 2,614 |
| Sector | 139 |
| State | 55 |
| Sector\_Group | 29 |
| Jobs\_Group | 14 |
| Revenue | 10 |
| Employee | 10 |
| Frecuency\_Salary | 5 |
| Profile | 4 |
| Remote | 3 |

**4. Rationale for Selecting One-Hot Encoding Threshold of 55**

* Features with a **manageable number of unique categories** can be one-hot encoded without causing a significant increase in dimensionality.
* **State** has **55 unique categories**, which is acceptable for one-hot encoding.
* Features like **Skills** with **10,805 unique categories** would drastically increase the number of features if one-hot encoded, so they are better handled differently (e.g., embedding or frequency encoding).
* **5. Compare Old and New PyCaret Model Performances**

**Old Parameters (One-Hot Encoding Threshold = 25)**

| **Model** | **RMSE** | **R2** | **MAE** | **MSE** | **RMSLE** | **MAPE** | **TT (Sec)** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **KNN Regressor** | **32,498.73** | **0.4494** | 22,596.23 | 1,057,256,761.60 | 0.2971 | 0.2359 | 0.1670 |
| Elastic Net | 35,664.40 | 0.3371 | 26,304.47 | 1,272,444,001.43 | 0.3382 | 0.2959 | 0.1190 |
| Extra Trees Regressor | 43,675.19 | 0.0058 | 32,972.50 | 1,907,943,506.80 | 0.4119 | 0.3746 | 1.0600 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| Bayesian Ridge | 99,749.11 | -5.1938 | 77,297.65 | 11,964,526,692.62 | 0.9147 | 0.8746 | 0.1890 |

**New Parameters (One-Hot Encoding Threshold = 55)**

| **Model** | **RMSE** | **R2** | **MAE** | **MSE** | **RMSLE** | **MAPE** | **TT (Sec)** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CatBoost Regressor** | **28,863.48** | **0.5655** | 20,557.89 | 834,343,765.99 | 0.2679 | 0.2174 | 0.9070 |
| Extreme Gradient Boosting | 28,999.66 | 0.5615 | 20,574.93 | 841,973,798.40 | 0.2696 | 0.2168 | 0.1380 |
| Random Forest Regressor | 29,030.88 | 0.5606 | 19,977.48 | 843,761,415.51 | 0.2681 | 0.2127 | 1.3570 |
| LightGBM Regressor | 29,649.35 | 0.5416 | 21,293.89 | 880,390,807.35 | 0.2760 | 0.2266 | 0.1620 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| Least Angle Regression | 43,043,992,652.23 | -8.13e12 | 15,104,356,656.34 | 1.58e22 | 4.2704 | 169,277.70 | 0.0580 |

**Key Differences**

* **Best Model Changed**: From **KNN Regressor** (Old) to **CatBoost Regressor** (New).
* **Improved RMSE**: Decreased from **32,498.73** to **28,863.48**.
* **Improved R²**: Increased from **0.4494** to **0.5655**.
* **Lower MAE**: Decreased from **22,596.23** to **20,557.89**.

**7. Comparative Analysis**

**Impact of Increasing One-Hot Encoding Threshold**

* **Feature Representation**: By increasing the one-hot encoding threshold from **25 to 55**, more categorical features were encoded, capturing more detailed information.
* **Model Performance**:
  + **RMSE Improved**: Indicates the model's predictions are closer to actual values.
  + **R² Increased**: Shows better explanatory power of the model.
* **Best Performing Model**:
  + **Old Parameters**: KNN Regressor performed best, likely due to the limited feature representation favoring instance-based learning.
  + **New Parameters**: CatBoost Regressor outperformed others, benefiting from the richer feature set and its ability to handle categorical variables effectively.

**Why RMSE is the Preferred Metric**

* **Punishes Larger Errors**: RMSE squares the errors before averaging, giving more weight to larger errors.
* **Sensitive to Outliers**: In salary prediction, large deviations can be significant, and RMSE captures this.
* **Focus on Prediction Accuracy**: Lower RMSE indicates the model predictions are, on average, very close to the actual salary values.

We ran into the error of not being able to apply the test set to our algorithm, because the training set was in it’s place and the code was not ready to be reused, otherwise it would just replace training data. So we decided to recreate the file using a pipeline.  
  
 **Issue:** Preprocessing logic was scattered, causing redundancy when switching between training and test datasets.

 **Solution:**

* Modularized preprocessing into a pipeline using sklearn to ensure consistency.
* Split the workflow into separate files for preprocessing, training, and inference.

Then we ran into more errors as we tried to create the pipeline: a .preprocessing.py file, train.py file and predict.py file:

**6. Mismatch Between Submission Format and Output**

* **Issue:** The generated submission.csv file didn’t match the required format of usjobs\_sample\_submission.
* **Solution:**
  + Extracted and preserved the ID column.
  + Created the submission file with only ID and Mean\_Salary columns.

Got decent results. But now realised skills is my 4th most important feature, I need to ohe this!! Then my metrics should be substantially better.

**6. Misaligned One-Hot Encoding Threshold**

**Issue Summary:**  
The Sector column, although important, was not included in one-hot encoding due to the threshold being too low.

**Solution:**

* Increased the one-hot encoding threshold to 139 to include Sector and other relevant categorical columns.
* Explicitly added Skills, Director, and City to the one-hot encoding pipeline.

**8. Excessive Features After One-Hot Encoding**

**Issue Summary:**  
One-hot encoding generated a large number of features, some of which were unimportant or irrelevant, leading to computational overhead.

**Solution:**

* Performed feature selection using feature importance scores from CatBoost.
* Retained only the top features contributing to 95% of cumulative importance.

**. Why are results worse?**

**a. Dimensionality reduction with PCA**

* PCA is used to reduce dimensionality by retaining only the most significant components of the data. By default, PCA reduces variance in the data but does not necessarily optimize for predictive performance. Important information could be lost in the process, especially when selecting an arbitrary number of components (e.g., 50 or 100). It’s possible that the PCA-transformed features are no longer strongly correlated with the target variable, leading to worse model performance.

**b. Increased complexity from one-hot encoding (OHE)**

* OHE increases the dimensionality of the dataset significantly (e.g., for Skills). Random Forest can handle high-dimensionality data, but excessive sparsity introduced by OHE might dilute the signal for meaningful predictors. If Skills features were not relevant to the target, their inclusion could introduce noise, decreasing performance.

**c. Feature interactions**

* By applying PCA after OHE, you may remove the interpretability of individual features and meaningful interactions between them. This could confuse tree-based models like Random Forest, which rely on clear feature splits.

**d. Overfitting to noise**

* Redundant preprocessing (e.g., frequency encoding for high-cardinality features combined with OHE or PCA) might increase noise or distort feature distributions, confusing the model.

Note: There was a too large a reliance on ChatGPT support toward the end of this project. I am aware that things got complex and I would like to ensure a smoother process in the future. I have learnt a tremendous amount about pipelines and simplicity through engaging with this project.

New Challenge:

Why my results were pretty bad was because I didn’T do a thorough enough EDA. Skills are multivalued, and ohe would only give me a combination of skills as binary 1 and 0 for no skills. I need to separate them out and take each unique skill and anaylse its impact on the mean salary

**Challenges with Skills**

Here, the Skills column is **not a simple categorical variable**. Each row can have multiple skills, e.g., "['Python', 'SQL', 'TensorFlow']". OHE does not natively handle such cases. If applied directly:

1. OHE would treat the **entire string representation of the skills** as a single category.
   * Example:

python

Copy code

Row 1: "['Python', 'SQL']" -> Column: ['Python', 'SQL'] = 1

Row 2: "['Python']" -> Column: ['Python'] = 1

* + - This results in columns corresponding to **unique combinations of skills**, which is not meaningful.

1. To apply OHE correctly, the skills would need to be "exploded" into individual rows or flattened into binary features (as done in your code).

Compare whether we need to flatten the columns, like for skills! We should havedone this a long time ago in eda we are going back to do it now.

ID: single

Job: multivalue

Jobs\_Group: single

Profile: single

Remote: single

Company: multivalue

Location: multivalue

City: single

State: single

Frecuency\_Salary: single

Mean\_Salary: single

Skills: multivalue

Sector: multivalue

Sector\_Group: single

Revenue: single

Employee: single

Company\_Score: single

Reviews: single

Director: multivalue

Director\_Score: single

URL: single

Please also bare in mind the original requirements

Your project aims to implement a machine learning workflow that includes a streamlined and consistent pipeline for preprocessing, training, and prediction. The preprocessing step should handle both training and test datasets to ensure that features are consistently encoded, aligned, and scaled. The training script should save the preprocessor, model, and expected feature names to ensure reproducibility. The prediction script should load these artifacts, align test data to the training schema, preprocess it using the preprocessing pipeline file, and generate predictions seamlessly. Robust error handling and logging should ensure clarity and traceability at every step, allowing for debugging and validation. Overall, the codebase should be modular, reusable, and flexible for future extensions, such as integrating new features or switching models. For encoding I would like ohe for all columns with 139 features or lower. For any high please do target encoding! The encoding should be done in the preprocessing pipeline. Here is the train data file. we are predicting mean salary. In the train.py file I want to use random forest! Please create the code for these 3 files!!

please help me scale the train and test sets through a function. Please ensure that all the numerical columns from each are scaled, except those columns which have 0-1, as these will be

ID: single

Job: multivalue

Jobs\_Group: single

Profile: single

Remote: single

Company: multivalue

Location: multivalue

City: single

State: single

Frecuency\_Salary: single

Mean\_Salary: single

Skills: multivalue

Sector: multivalue

Sector\_Group: single

Revenue: single

Employee: single

Company\_Score: single

Reviews: single

Director: multivalue

Director\_Score: single

URL: single

Unique values in multivalued columns:

Skills: 99

Location: 12737

Sector: 147

Company: 14145

Job: 18028

Director: 2718

|  |  |
| --- | --- |
| Sector | 139 |
| State | 55 |
| Sector\_Group | 29 |
| Jobs\_Group | 14 |
| Revenue | 10 |
| Employee | 10 |
| Frecuency\_Salary | 5 |
| Profile | 4 |
| Remote | 3 |

To include and add to below:

Two code files journeys and their differences:

Final code pipeline:

Preprocessing pipeline, training pipeline and predict pipeline. I frequency encoded, and 139 ohe threshold for unique values. I StandardScaled as minmax gave me worse results. I median imputed as mean gave me worse results. Catboost performed best here.

Jupiter notebook:

I target encoded values above, and 139 ohe threshold for unique values. I minmaxscaled (to have the same similar values as 0 and 1 that I had for other values like my ohe values. Gradientboosting regressor performed best here.

Time line:

So we proceeded first in Jupyter notebook, from beginning to end with Pycaret. Afterward I built a pipeline as I realised it was tricky to also preprocess the test data like the training data. However that took a lot of time and learning. That worked well, but with how complex it got, I wanted to then go back to jupyter notebook to encode and scale the data with more finesse, and potentially also remove outliers. This went well for a bit but then after a lot of time spent on it, with a longer and more complex good model. The model still performed worse than my original... I am still not sure why. I attempted to split ohe at 139 unique values, which is a good decision. I did target encoding for those above that instead of frequency encoding like I now have in my pipeline. Overall the code was challenging to implement, but am I very proud of how much I learnt through this project. In every area I enriched my learnings, code and ML strategies and organisation.

In the future my main takes aways are: Stick to one way, pipeline or notebook! More EDA, look at your data shape and columns and rows and values after each step. It will be worth not going back and changing a bunch of stuff, and instead making intelligent decisions as you go. Follow a step by step procedure more closely, and not try to see the end result so soon. It is tempting to be quicker and just see how your model performs, but if you understand each part you have done, then you will probably have a good model at the end and you can best change any section if you want to refine.

Please integrate the above text into the draft report, and shape the draft report around these findings more. Please keep syntax similar to the above text throughout the whole draft report. Similarly casual and honest, and written personally, using more active sentences with ‘I’