Intro to ML - Assignment 1

Jaffar Totanji - j.totanji@innopolis.university

Theoretical Part:

2.1 Regarding the Preprocessing:

- **Which regression model was the most effective for the missing values, and why?**
  - I found `Linear Regression` to be the most effective model for predicting the missing values.
  - Reasoning: After encoding the values of the columns containing categorical values, 2 models were implemented: `Linear Regression`, and `Polynomial Regression` of degrees [2, 3, 4].

    `Polynomial Regression` of degree 2 had the lowest MSE out of all the others when applied to the testing data.

    However, when calculating the Cross-Validation score of the previous methods using `K-Fold` technique, we notice that `Linear Regression` maintains the lowest MSE.

    The reason for that being that `Polynomial Regression` was actually `overfitting` our training data, and through `Cross-Validation`, we were able to see the errors it was making on the testing data when it met more unseen data.

    We can see that more clearly as the cross-validated MSE for `Polynomial Regression` becomes significantly higher the higher the degree.

    Our data is too simple for `Polynomial Regression` and thus `Linear Regression` is what fits it best.
- **What encoding technique did you use for encoding the categorical features, and why?**
  - I used `Ordinal Encoding` for `var3`, and `One-Hot Encoding` for `var6`.
  - Reasoning: I chose to trade accuracy for efficiency here.

    It is true that `Ordinal Encoding` is used for categorical values which have a natural ranking order unlike `var3`, but using `One-Hot Encoding` for the name values in `var3` would result in too many extra columns which is neither efficient nor proportional to the size of the dataset.

    For `var6` I used be using `One-Hot Encoding`, as there is no natural ranking order between the values it can take, and also because the column can only take one of two values, so `One-Hot Encoding` would only result in one extra column which is quite good in terms of efficiency.

2.2 Regarding the training process:

- **Which classification model performed best, and why?**

- `KNN` performed best.
- Reasoning:

  The dataset and circumstances met the conditions for `KNN`s success.

  First, `Logistic Regression` works well with a Linearly Seperable dataset and not very well with a non-linear classification problem. We saw from our visualization that our dataset wasn't exactly linear, hence, `Logistic Regression` didn't do a very good job.

  `KNN` and `Naive Bayes` both work well with both linear and non-linear classification problems, our case is the latter.

  Second, `Naive Bayes` expects features to be strictly independent to each other, which is not exactly applicable in real-life scenarios. Our dataset contained featrues that weren't exacty independent to each other, as we were able to collect ~90% of the dataset's variance in only 2 vectors when using `PCA`. Indicating that our features share some a relation.

  That along with the fact that `KNN` works well best with properly-scaled features (which we did by using StandardizedScaler), leave `KNN` with no conditions contributing to its failure, unlike the other 2 models.

  Hence, `KNN` had the best performance.

- **What were the most critical features with regards to the classification, and why?**
  - The most significant features were `var1`, `var4`, and possibly `var5` and `var2` (The exact values of their coefficients are printed in the `Logistic Regression` section in the code).
  - Reasoning: We can determine what the most significant features are by looking at the absolute value of their coefficients in our `Logistic Regression` model.

  The higher the value, the more impact this feature has on our accuracy, and these 4 had the highest values overall, with the first 2 being the highest.

- **What features might be redundant or are not useful, and why?**
  - The features that might be considered redundant are `var3` and `var6`.
  - Reasoning: Same as the previous question. These 2 had the lowest coefficients overall.

- **Did the dimensionality reduction by the PCA improve the model performance, and why?**
  - No, in fact, it made it worse.
  - Reasoning: `PCA` is used to reduce the dimesnsionality of the dataset.

    It does that by finding the best linear combinations of the original features of the dataset, and using them as new features of the dataset.

    It allows us to evaluate the resulting vectors (linear combinations) and choose the best ones. The `best` ones are basically the ones that best represent our original dataset i.e. the ones that contain most of the variance in our dataset to best represent it.

I tried choosing both 2 and 3 vectors represnting ~78.2% and ~90.9% of the variance in the original dataset. They produced worse results than the original features when used with the models.

That's simply because 90% is still less than a 100% (what we got from using our original features), and to reach ~100% we would have to work with 5-7 vectors. Which is around the same number as the original features in our dataset, so we're better off using the original ones instead.

PCA was useful here but only for visualization. It would be useful for training models in a scenario where most of the variance can be contained in a number of vectors that is significantly less than the original features of the dataset.