Binary Classification

Assignment

Background: Imagine that you are an admissions officer for a highly sought-after graduate program. As part of a new initiative to streamline the admissions process, a function – whose precise specifications are known only to the administration – has been developed to evaluate each application with respect to some criteria. A composite metric known as 'fit' is computed for each individual candidate; those scoring *greater than or equal to 70* are extended an admissions offer.

Objective: In the absence of a clear-cut function with which to assign 'fit' scores, learn a classification rule by assembling and training classification models on historical admissions data.

Note: For reproducible results, include `random_state=42` as a parameter to Sklearn functions.

1. **Preprocessing and Exploratory Analytics:**

   - Examine the structure and format of the data. Is it suitable for logistic regression? How do the features relate to one another? If necessary, encode the target variable.
   - Apply PCA to the data and select the first two principal components that *account for 80% of the variance*. We are performing dimensionality reduction merely for the sake of visualization – in this instance, note the adverse effect that reduction of variance has on the model's overall performance.
   - Split the data into a training and testing set. The testing set should be one fifth the size of the training set.

2. **Model Assembly and Evaluation:**

   - Implement the following classification models: **(1)** *logistic regression*, **(2)** *linear support vector classification*, **(3)** *k-nearest neighbors classification*, and **(4)** *decision tree classification*.
   - Train and test your classifiers using the previously reduced data. For each model, plot the classification probability.
   - Manually tune your models' hyperparameters so that each achieves a *Cohen's kappa statistic* greater than 0.65. (View Cohen 1960 for further information on the kappa performance metric.)
     - For logistic regression and support vector classification, adjust the inverse regularization parameter `C` and observe how the probability threshold reacts. Considering the ratio between accepted and rejected applicants, adjust the `class_weight` parameter accordingly.
     - For k-nearest neighbors classification, consider adjusting the `n_neighbors` parameter. Generally, the empirically optimal K is found using an optimization heuristic – in this case, adjust the parameter manually and observe any changes.
   - Compute the *accuracy* score for each classifier. Why might this be a misleading performance metric?
   - Shuffle your data set. If at all, how does this affect your results? Do decision trees differ between runs? For interpretability, use the PyDotPlus package to visualize your decision trees.