Solution to Examination: Programming for Data Science (ID2214)

Henrik Boström

January 9, 2020

Part I (Theory, 10 points)

1a. Methodology, 2 points

Since we have evaluated 100 different configurations for the novel algorithm, the observed (cross-validation) accuracy for the best performing of these is most likely biased, i.e., the performance is over-estimated, due to sampling error. Hence, although the best performing configuration outperforms the baseline when performing cross-validation on the first half of the data, the corresponding model trained with this configuration on this half and evaluated on the second half may very well be outperformed by the baseline. However, if a majority of the evaluated configurations outperform the baseline on the first half, we may expect the best performing configuration to still outperform the baseline on the second half.

1b. Data preparation, 2 points

Min-max normalization preserves the order of the values, i.e., if a value has rank r before normalization, the transformed value will also have the rank r. This means that the normalization has no effect when binning with equal-sized bins, i.e., bins with equal frequencies of observed values, in the sence that any numerical feature value will be replaced by the same (categorical) bin name, indepedently of whether all values have been normalized or not. This assumes that only the bin names are used during model building and not their definitions.

1c. Performance metrics, 2 points

Since the model has a higher precision on the majority class instances, and the relative frequency of these decrease when balancing the test set, we should expect to see a decreased accuracy. For AUC, we should expect to see about the same performance, since this metric concerns the model's ability to rank a positive instance ahead of a negative, which is not affected in any direction by changing the class proportions in the test set. However, the variability may increase due to having a smaller sample size.

1d. Combining models, 2 points

Random forests form their predictions by averaging the predictions of the individual trees, and since these (normally) are restricted to making predictions by averaging observed regression values in the leaves, they will not produce predictions that are higher (lower) than the highest (lowest) regression value in the training set. Gradient boosting machines form their predictions by summing the predictions of the individual trees, which may very well result in predictions that fall outside the range of observed regression values.

1e. Clustering, 2 points

Ties appear when performing agglomerative (bottom-up) clustering when two or more alternatives of clusters to merge result in the same score. In general, resolving such ties randomly, i.e., picking any of the alternatives arbitrarily with some element of chance, could result in completely different clusterings, and would hence motivate multiple re-runs, similar to what is recommended for k-means clustering. (If no such ties occur, then there is no point to re-run agglomerative clustering, as it becomes completely deterministic, which is also the case if we choose to handle ties in a non-stochastic way). When single-linkage is used, the score is the smallest distance between a pair of elements in two different clusters. This means that in case of ties between several alternative clusters to merge, these will by necessity be merged in sequence; the merging of two clusters cannot result in that the smallest distance between any pair of clusters becomes smaller than for the tied clusters. Since the clusters are merged in sequence, the exact order in which this is done has no effect on sub-sequent mergings.

Part II (Programming, 20 points)

2a. Data preparation, 10 points

```
def bag_of_words(word_lists):
    column_index = 0
    mapping = {}
    for paragraph in word_lists:
        for word in paragraph:
            if mapping.get(word) is None:
                mapping[word] = column_index
                column_index += 1
    matrix = np.zeros((len(word_lists),column_index),dtype=int)
    for (row,paragraph) in enumerate(word_lists):
        for word in paragraph:
            matrix[row,mapping[word]] = 1
    return (mapping,matrix)
```

2b. Combining models, 10 points

```
def variable_importance(df,model):
    orig_predictions = model.predict(df)
    correct_labels = df["CLASS"]
    orig_acc = accuracy(orig_predictions,correct_labels)
    raw_var_imp = [(feature,max(0,orig_acc-perm_acc(df,feature,correct_labels,model)))
                   for feature in df.columns if feature != "CLASS"]
    sum_imp = sum([fi[1] for fi in raw_var_imp])
    normalized_var_imp = [(fi[0],fi[1]/sum_imp) for fi in raw_var_imp]
    return normalized_var_imp

def perm_acc(df,feature,correct_labels,model):
    orig_values = df[feature].copy()
    df[feature] = np.random.permutation(orig_values)
    perm_accuracy = accuracy(model.predict(df),correct_labels)
    df[feature] = orig_values
    return perm_accuracy
```