Solution to Examination: Programming for Data Science (ID2214)

Henrik Boström

April 14, 2020

Part I (Theory, 10 points)

1a. Methodology, 2 points

To obtain an estimate of the performance on independent data, we need to make sure that no information from the test set is carried over to the trained model. However, if we employ imputation before splitting the dataset into a training and a test set, some information from the test set may be used when imputing feature values in the training set. This could potentially lead to that the model is better fitted to the test set than to some other independent test set, from which no information has been extracted.

1b. Data preparation, 2 points

If equal-width binning is employed, it may result in that the training instances are not distributed uniformly among the child nodes, when splitting a node during tree growth with the discretized feature. In the extreme case, this could mean that few instances are separated out, reducing the likelihood of the feature being selected, and increasing tree depth, if the feature is indeed selected. If equal-sized binning is employed, the instances will be distributed uniformly over the child nodes, typically leading to shallower trees, if the feature is selected.

1c. Performance metrics, 2 points

The precision for the positive class is estimated by the number of predictions for the positive class that are correct divided by the number of predictions for the positive class. Assuming that the class probability estimate is positively correlated with the probability of class membership, the precision will increase when increasing the threshold, as the probability of being correctly classified increases.

If we decrease the threshold, more instances will be classified as positive, which means that we can expect a larger fraction of the instances that belong to the positive class to be classified as positive, which hence means that the recall is expected to increase when we decrease the threshold.

1d. Combining models, 2 points

If we would like to generate an ensemble of naïve Bayes classifiers using the strategy of random forests, it would mean that we introduce diversity in two ways; by training each classifier from a bootstrap replicate (through bagging) and by considering a random subset of the features for each classifier.

1e. Association rules, 2 points

It may be that multiple rules are applicable, i.e., the conditions (antecedents) are subsets of the itemset representing the instance to be classified, but with different consequents; this means that the rules are in conflict regarding what class label to assign.

It may be that for some test instance, there is no rule such that the condition (antecedent) is a subset of the itemset representing the instance to be classified; this means that there is no applicable rule that can suggest what class label to assign.

Part II (Programming, 20 points)

2a. Data preparation, 10 points

```
def aggregate(df):
    unique_ids = df["ID"].unique()
    new_df = pd.DataFrame()
    for col in df.columns:
        if col == "ID":
            new_df["ID"] = unique_ids
        elif col == "CLASS":
            new_df[col] = [df.loc[df["ID"]==i,col].mode()[0]
                           for i in unique_ids]
        else:
            new_df[col] = [df.loc[df["ID"]==i,col].mean()
                           for i in unique_ids]
    return new_df

def aggregate_faster(df): # faster version
    unique_ids = df["ID"].unique()
    indexes = [np.nonzero(df["ID"]==i)[0] for i in unique_ids]
    new_df = pd.DataFrame()
    for col_num, col in enumerate(df.columns):
        if col == "ID":
            new_df["ID"] = unique_ids
        elif col == "CLASS":
            new_df[col] = [df.iloc[indexes[i],col_num].mode()[0]
                           for i in range(len(unique_ids))]
        else:
            new_df[col] = [df.iloc[indexes[i],col_num].mean()
                           for i in range(len(unique_ids))]
    return new_df
```

2b. Combining models, 10 points

```
def stacking(level0_df,level1_df,base_learners,learner):
    base_models = [alg.fit(level0_df) for alg in base_learners]
    new_df = pd.DataFrame()
    new_df["CLASS"] = level1_df["CLASS"]
    for i, model in enumerate(base_models):
        new_df[i] = model.predict(level1_df)
    stacking_model = learner.fit(new_df)
    return base_models, stacking_model
```