Machine Learning WS 19 - Assignment 6

November 29, 2018

Adrian Gruszczynski / Yann Salimi

```
In [23]: import numpy as np
         import pandas as pd
```

1 Decision tree

```
In [3]: class DTNode:
            def __init__(self, feature, threshold):
                self.feature = feature
                self.threshold = threshold
                self.left = None
                self.right = None

            def predict(self, x):
                if not self._is_initialized:
                    raise ValueError('node is not initialized')
                if x[self.feature] < self.threshold:
                    return self.left.predict(x)
                else:
                    return self.right.predict(x)

            @property
            def _is_initialized(self):
                return self.left and self.right


        class DTLeaf:
            def __init__(self, y):
                self.y = y

            def predict(self, _):
                return self.y

In [10]: def entropy(X):
             probabilities = np.bincount(X) / len(X)
             probabilities = probabilities[probabilities > 0]
```

```
    return -np.sum(probabilities * np.log2(probabilities))


def buildDT(X, y):
    best_information_gain, node_data = 0, None
    n_samples, n_features = X.shape
    H_before_split = entropy(y)

    for feature in range(n_features):
        X_feature = X[:, feature]
        threshold = np.mean(X_feature)
        left_idx = X_feature < threshold
        right_idx = X_feature >= threshold
        y_left = y[left_idx]
        y_right = y[right_idx]
        p_y_left = len(y_left) / n_samples
        p_y_right = len(y_right) / n_samples
        H_after_split = p_y_left * entropy(y_left) + p_y_right * entropy(y_right)
        information_gain = H_before_split - H_after_split

        if information_gain > best_information_gain:
            best_information_gain = information_gain
            node_data = feature, threshold, left_idx, y_left, right_idx, y_right

    if not best_information_gain:
        return DTLeaf(y[0])
    else:
        feature, threshold, left_idx, y_left, right_idx, y_right = node_data
        node = DTNode(feature, threshold)
        node.left = buildDT(X[left_idx], y_left)
        node.right = buildDT(X[right_idx], y_right)
        return node


def unison_shuffle(a, b):
    if len(a) != len(b):
        raise ValueError('array lengths do not match')
    idx = np.random.permutation(len(a))
    return a[idx], b[idx]


def accuracy_score(y_true, y_pred):
    if y_true.shape != y_pred.shape:
        raise ValueError('array shapes do not match')
    return np.sum(np.equal(y_true, y_pred)) / len(y_true)


def confusion_matrix(y_true, y_pred):
```

```
            y_actual = pd.Series(y_true, name='Actual')
            y_pred = pd.Series(y_pred, name='Predicted')
            return pd.crosstab(y_actual, y_pred,
                               rownames=['Actual'],
                               colnames=['Predicted'],
                               margins=True)
```

```
In [12]: np.random.seed(12345)
         df = np.array(pd.read_csv('spambase.data', header=None))

         X, y = df[:, :-1], df[:, -1].astype(np.bool_)
         X, y = unison_shuffle(X, y)

         split = len(X) // 2

         X_train, y_train = X[:split], y[:split]
         X_val, y_val = X[split:], y[split:]
```

```
In [8]: decision_tree = buildDT(X_train, y_train)
```

```
In [11]: y_pred = np.empty(y_val.shape)
         for i in range(len(X_val)):
             y_pred[i] = decision_tree.predict(X_val[i])

         conf_m_tree = confusion_matrix(y_train, y_pred)
         accuracy = 100 * accuracy_score(y_val, y_pred)
         print(conf_m_tree)
         print('decision tree accuracy: %.2f%%' % accuracy)
```

```
Predicted   0.0  1.0   All
Actual
False       849  533  1382
True        574  344   918
All        1423  877  2300
decision tree accuracy: 90.53%
```

1.1 Assume that classifying a genuine E-Mail as spam is ten times worse than classifying spam as genuine. How would you change the design of your decision tree?

In order to fullfill the assumption's requirement pruning would be necessary.

Predictions shall be only positive if the certainty, that a given sample is spam is sufficient.

For this a vote mechanism could be implemented, that classifies a sample positive only if 90% of labels are positive.

1.2 Can Information Gain be negative? Try to prove your answer.

1.2.1 Decision tree context

In the context of the decision trees the information gain is defined as:

$IG(Y|X) = H(Y) - H(Y|X)$
Whereas $H(Y)$ is the entropy of the parent node and $H(Y|X)$ the entropy of the split.
Assume worst case scenario in which node Y cannnot be splitted anymore.
Then the enrtopy $H(Y)$ equals this of $H(Y|X)$
It holds that $H(Y) = H(Y|X) - H(Y|X) >= 0$

1.2.2 Formally

From the Gibb's inequality it follows that:
$D_{KL}(P||Q) = \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \geq 0$
The only case $D_{KL} = 0$ is when $P = Q$
As a result, the information gain can not be negative

2 Random forest

```
In [17]: class DTForest:
             def __init__(self, decision_trees):
                 self.decision_trees = decision_trees

             def predict(self, x):
                 y_pred = [dt.predict(x) for dt in self.decision_trees]
                 most_frequent_y = np.argmax(np.bincount(y_pred))
                 return most_frequent_y
```

```
In [15]: forest_size = 5
         decision_trees = []
```

```
In [18]: for tree_idx in range(forest_size):
             sampled_idx = np.random.randint(0, high=split, size=split)
             X_bootstrap, y_bootstrap = X_train[sampled_idx], y_train[sampled_idx]
             decision_tree = buildDT(X_bootstrap, y_bootstrap)
             decision_trees.append(decision_tree)

         decision_forest = DTForest(decision_trees)
```

```
In [19]: y_pred = np.empty(y_val.shape)
         for i in range(len(X_val)):
             y_pred[i] = decision_forest.predict(X_val[i])
```

```
In [22]: conf_m_tree = confusion_matrix(y_train, y_pred)
         accuracy = 100 * accuracy_score(y_val, y_pred)
         print(conf_m_tree)
         print('forest accuracy: %.2f%%' % accuracy)
```

```
Predicted   0.0  1.0   All
Actual
False       848  534  1382
True        583  335   918
```

```
All         1431  869  2300
forest accuracy: 93.83%
```

2.1 What is a good number of trees in the forest?

A huge increase in number of trees in the forest does not boost the accuracy appropriately.

In our case the difference in accuracy between a forest with 5 and a forest with 100 trees equals 0.2%

2.2 What is the best way to decide?

The best way to decide would be to consider the probability that the given sample is spam.

Probability higher than a certain threshold could be classfied as positive otherwise as negative.