CS57800: Statistical Machine Learning

Homework 1

Ruoyu Wu
wu1377@purdue.edu

September 19, 2018

1 Code

1.1 K-fold validation

Implemented in k_cross.py.

Nested loop with splitting data(both feature and label), fit() and predict(). This module also prints out the necessary info for the report. The script with sufficient comments should clearly explains itself.

1.2 Metric

Implemented in metric.py

The metrics, including *precision*, *recall*, *f1 score* and *accuracy* are tested by comparing the results with the ones in sklearn library.

1.3 KNN

Implemented in knn.py.

One way of generalizing distance is through minkowski distance.

$$D(X,Y) = (\Sigma_{i=1}^{n}|x_i - y_i|^p)^{1/p}$$

It is typically used used with p being 1 or 2, which correspond to the Manhattan distance and Euclidean distance.

And another metric I implemented is cosine distance.

$$D(X,Y) = \frac{X * Y}{\sqrt{X * X}\sqrt{Y * Y}}$$

This implementation use naive way to search for neighbors without any optimization methods.

1.4 Decision Tree

Implemented in decision_tree.py.

Implement a naive ID3 decision tree with information gain as splitting metric. Except for max_depth, there are a few more parameters to tune. For example, min_sample_split, min_impunity_decrease. These will be explained in the Bonus Point.

The script with sufficient comments should clearly explain itself.

1.5 Scope of Creativity

I implemented two ways of finding the threshold to categorize the continuous feature.

- During the training process, for each candidate feature, we first find the range of the feature, i.e. find the max and min value of that feature in the training sample, split it equally into 10 chunks, and use them as split threshold. This thought came to me before seeing the normalization using sigmoid, so it is my default method. The disadvantage is obvious, the range of that feature on training set might not equal to the one on testing set(though the expectation is the same).
- Another one is using sigmoid function to map each continuous feature into [0, 1], and use range(0.1, 1, 0.1) as split threshold. This approach normalizes the value and I expect it a better performance.

Therefore, in the section of decision tree report, I have two outcomes, one without sigmoid and one with sigmoid.

2 Report

2.1 Clarification

The dataset turns out to be very imbalance. For the whole dataset, label '5' and '6' dominate, they together take 3655 instances out of 4898. And label '3' only has 20 instances, label '9' only has 5. So if we use naive KNN or DT(without weighted method), the classifier definitely will tend to predict the value of majority class for all predictions and achieve a high accuracy.

Actually, as you can see from section 2.3, the classifiers from third-party library(sklearn) also do not perform well without using any weighting methods. Their performance are even worse than predicting all the instances as label '5', which occurs 2198 out of 4898 in the whole dataset, that is 0.448755 accuracy.

In the specification of this homework, we are required to "report the best preforming one". But there are two metrics, and they always grow in opposite direction. Because there are no details about this 'task', i.e. how important the minority class is, I decide to report the output with higher accuracy, without degrading the f1 score too much.

2.2 Performance

2.2.1 KNN

For KNN, we have two parameter to tune, k and metric. k is the number of nearest neighbors we use to predict the label of a instance. Metric is for us to choose the type of distance we want to use. When metric='cosine', we use cosine distance. When metric is set to 'minkowski', p=1 means Manhattan distance and p=2 means Euclidean distance.

Through experiments from all the pairs of parameters, we find that the best performance comes when K = 4 and using Manhattan distance, with average accuracy 0.415645 and average f1 score 0.238368. The result is following:

Fold	validation accuracy	validation f1	test accuracy	test f1
Fold 1	0.425654	0.270604	0.378268	0.182716
Fold 2	0.406318	0.231944	0.429739	0.256293
Fold 3	0.415577	0.237037	0.428922	0.226757
Fold 4	0.407680	0.230191	0.425654	0.287708

As I observe from the trend of K vs. performance, I find that the larger the K is, the better the performance, especially the accuracy. I think it is because of the imbalance nature of the dataset. The larger the K is set, this classifier more tends to predict all the instance as the major class label. And by the way, all the accuracy above are lower than the major label's precentage in the dataset.

2.2.2 Decision Tree(without sigmoid)

The best performance(only considering accuracy) occurs when max_depth = 5, with 0.449755 average accuracy and 0.211307 average f1 score. The result is following:

Fold	training accuracy	training f1	validation accuracy	validation f1	test accuracy	test f1
Fold 1	0.469612	0.182868	0.466776	0.291085	0.417484	0.180843
Fold 2	0.459361	0.203595	0.452342	0.262147	0.434641	0.169951
Fold 3	0.458636	0.229342	0.449074	0.272317	0.455065	0.236176
Fold 4	0.439224	0.214990	0.437092	0.261591	0.491830	0.258256

When considering both the accuracy and f1 score, however, the best performance occurs when max_depth = 1, which predict all the instance as label '5', with average accuracy 0.448734 and average f1 score 0.618674.

2.2.3 Decision Tree(with sigmoid)

The best performance(considering both accuracy and f1 score) occurs when max_depth = 2, with 0.450776 average accuracy and 0.327824 average f1 score. The result is following:

Fold	training accuracy	training f1	validation accuracy	validation f1	test accuracy	test f1
Fold 1	0.462990	0.482901	0.462963	0.482663	0.411765	0.308698
Fold 2	0.454826	0.329742	0.454793	0.328700	0.438725	0.322933
Fold 3	0.451560	0.328260	0.451525	0.327753	0.448529	0.328093
Fold 4	0.429517	0.401597	0.432190	0.397352	0.504085	0.351573

As expected, decision tree with sigmoid function as normalization achieves better performance than the one without.

2.3 Comparing with Third-Party Toolkits

I found my result is not goof, so I tried the KNN classifier and decision tree classifier from sklearn(default setting), and the result is following:

Classifier	test accuracy	test f1
KNN	0.350369	0.200927
Decision Tree	0.396207	0.201753

From the result of the third library, we can see the sklearn learn in default setting is even worse than mine, for both KNN and DT, in the sense of both accuracy and f1 score.

2.4 DT Graph: Max-depth vs performance

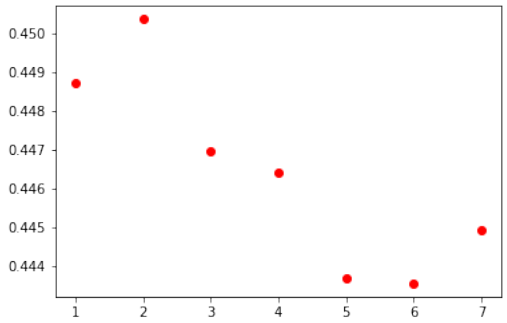

Figure 1: max_depth vs accuracy on validation

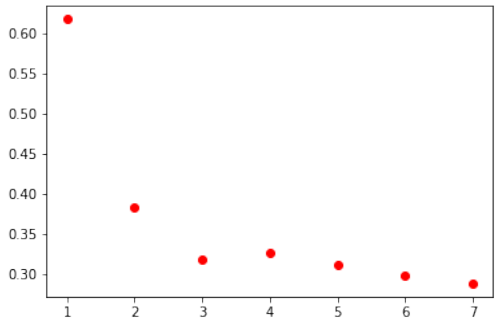

Figure 2: max_depth vs f1 on validation

2.5 KNN Graph: Distance metrics and K vs performance

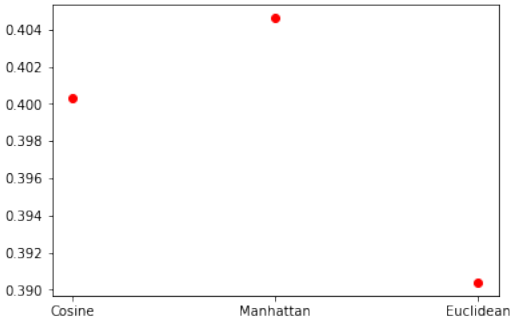

Figure 3: distance metrics vs accuracy on validation

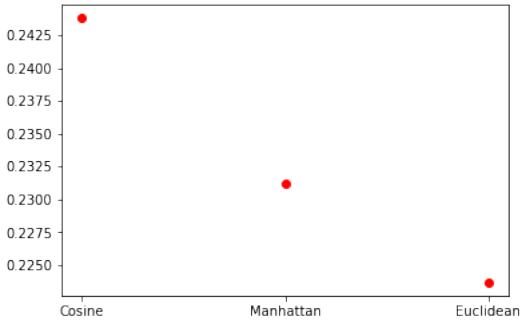

Figure 4: distance metrics vs f1 on validation

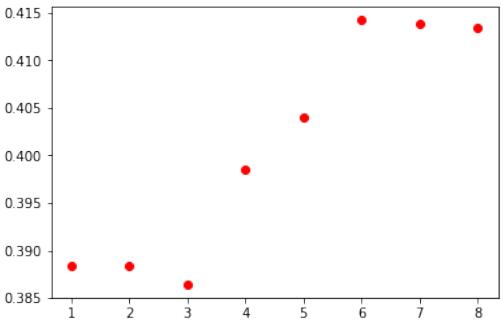

Figure 5: K vs accuracy on validation

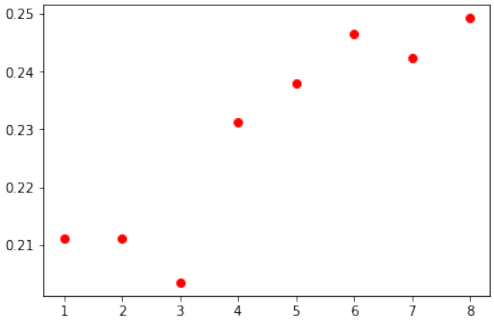

Figure 6: K vs f1 on validation

2.6 Answer the questions

2.6.1 1

If the same features implying the same label in the training instances, it means that this decision tree can achieve perfect performance on training set, while it is over-fitting and might be awful on the testing set.

2.6.2 2

The decision tree classifier build up a model upon training set, and only then they can do prediction on test set. KNN classifier, however, starts processing data only after it is given a test

set.

2.6.3 3

Using regression decision tree. When splitting, we can use least squares(to the average target value) as metric(to minimize it). And in the leaf node, instead of having class label, we use the average target value. Then We can rank by the value our regression tree gives.

3 Bonus Points

3.1 Best Result

The best result comes with the decision tree(max_depth=2 and with sigmoid normalization), 0.450776 average accuracy and 0.327824 average f1 score.

3.2 Anything Else

3.2.1 Minimun information decrease

In the decision tree algorithm, one way to prevent over-fitting is to impose more strict condition upon the splitting of the node. Our algorithm splits the node based on information gain. We can set a *minimun information gain*. If the maximum information gain of splitting the current node is below this threshold, then the splitting ends and the current node becomes a leaf.

From the experiments(setting the min_impurity_decrease to 1.0 and 0.5), it does not outperform the decision tree with sigmoid function, I believe it is because of the imbalance dataset.

3.2.2 Minimum sample split

Minimum sample split is the minimum number of samples required to split an internal node. It prevents little instances split the node, enhances the generalization ability.