Amin Fadaee

Machine Learning HW2

Python 3.5, Libraries: Numpy, Pandas, Random, Documentation: HTML, Bootstrap 4, jqMath

KNN Algorithm

Problem 1

KNN Algorithm needs sufficient data in order to have a reasonably good result and in times when there are not enough data points, the k-nearest neighbor for a test data deviate tremendously with the true label. By adding new dimensions to the data, the need for new point increase exponentially and sparse high dimension data would make KNN a really poor choice for a classifier.

Problem 2

The bias and variance of KNN algorithm is highly dependant on the **K** parameter. This relationship can be stated in the following way:

- For $k = \infty$ We have the maximum bias due to the fact that every test data would be classified as the class with highest prior probability; In the other hand this would make the variance of the model as low as possible and if we suppose that the prior probability is fixed the variance would be actually zero for different sets of data points.
- In the case of $k = 1$ we have the maximum variance, this is due to the fact that in this case the decision boundaries create a voronoi cell network which is highly dependent on the positions of the data points and a new set of data would completely change the boundary.

Problem 3

Based on *Murphy*'s definition in his book **Machine Learning, a Probabilistic Approach**:

> Does the model have a fixed number of parameters, or does the number of parameters grow with the amount of training data? The former is called a parametric model, and the latter is called a nonparametric model. Parametric models have the advantage of often being faster to use, but the disadvantage of making stronger assumptions about the nature of the data distributions. Nonparametric models are more flexible, but often computationally intractable for large datasets.

This definition is exactly correspondent to what KNN does and how it behaves.

Problem 4

a. Using Euclidean Distance, 10-Fold cross validation has been done to find the best K for classifying the **Seeds Data** (http://archive.ics.uci.edu/ml/datasets/seeds) among options of $K \in 1, 3, 5, 7, 10$. After performing the cross validation the following results were obtained

K	Accuracy
1	0.9142
3	0.8809
5	0.8857
7	0.8904
10	0.9000

and K=1 is the best choice for further classifications. The following functions were used in obtaining these results in `KNN4.py` :

- `(data, labels, options, distance=euclidean_distance)` : By performing 10-Fold Cross Validation, finds the optimal k for KNN
- `KNN(data, labels, test, k, distance=euclidean_distance)` : Computes the KNN classification on test with the provided distance function

---

b. 10-Fold cross validation was again used to assess the performance of KNN with $K = 5$ with the following distance measures:

- Euclidean
- Manhattan
- Minkowski(4)
- Minkowski($\frac{1}{2}$)
- Cosine

And here is the results:

Distance Measure	Accuracy
Euclidean	0.8857
Manhattan	0.9000
Minkowski(4)	0.9000
Minkowski($\frac{1}{2}$)	0.9000
Cosine	0.9095

As we can see, for K=5 the Cosine Distance obtains the best accuracy on the data. The following functions were used in obtaining these results in `KNN4.py` :

- `euclidean_distance(p1, p2)` : Computes the euclidean distance between p1 and p2
- `manhattan_distance(p1, p2)` : Computes the manhattan distance between p1 and p2
- `cosine_distance(p1, p2)` : Computes the cosine distance between p1 and p2
- `minkowski_distance(p1, p2, p)` : Computes the minkowski distance between p1 and p2 with parameter p
- `KNN(data, labels, test, k, distance=euclidean_distance)` : Computes the KNN classification on test with the provided distance function
- `cross_validation_distances(data, labels, distance_functions, k=5)` : By performing 10-Fold Cross Validation, finds the test error of KNN for different distance measures.

Problem 5

a. Based on the paper KNN shines for its simplicity and scalability, however it does not perform well in on imbalanced data.
b. Some ways were mentioned in dealing with imbalanced data, such as under-sampling of the major class instances or over-sampling for the minority class or changing the probability density distribution by setting different weights for different classes.
c. The WDKNN improves better in some cases because it assigns different weights on different nearest neighbors in a way that the weight has a reverse linear relationship with the point distance with the test data. The WDKNN algorithm has been implemented in `KNN5.py` and the results will be discussed in part e.
d. The paper deals with the imbalanced data by increasing the weight for the unsafe minority instances. This increase is determined by the number of majority instances that exist in that point's K nearest neighborhood
e. First the 7 original data sets were undergone some basic pre-processings:
   1. Meta-data in the beginning of the files were removed
   2. A header line was added with the names of each column
   3. The last column name is *Class*
   4. *Positive* was changed to *1* and *Negative* to *0*
   5. Files format was changed to `.csv`

   The following functions were implemented and used:
   1. `distance(p1, p2)` : Computes the euclidean distance between p1 and p2
   2. `find_KNN(data, labels, test, k, test_in_data=False)` : Computes the k nearest neighbor of test.
   3. `WDKNN(data, labels, test, k, test_in_data=False)` : Computes the WDKNN classification on test with the provided distance function
   4. `MCKNN(data, labels, test, k, minority_label, test_in_data=False)` : Computes the Minority Class KNN classification on test with the provided distance function
   5. `five_fold(data, labels, k, function)` : Performs the 5-Fold Cross Validation of the above algorithms and returns G-Mean and F-Measure of the classification
   6. `precision(TP, FP)`
   7. `recall(TP, FN)`
   8. `g_mean(TP, TN, FP, FN)`
   9. `f_measure(TP, TN, FP, FN)`

   After performing both algorithms on all the 7 data sets, the G-Mean and F-Measure of each set was obtained and can be seen in the following table:

	WDKNN		MCKNN	
Data	F-Measure	G-Mean	F-Measure	G-Mean
yeast3	0.7357	0.8133	0.7530	0.8522
ecoli3	0.6470	0.7782	0.6666	0.8371
yeast-2_vs_4	0.7294	0.7771	0.7032	0.7852
yeast-0-3-5-9_vs_7-8	0.3733	0.5227	0.3838	0.5958
yeast-0-2-5-6_vs_3-7-8-9	0.6341	0.7195	0.6192	0.7687
yeast-0-2-5-7-9_vs_3-6-8	0.8148	0.8755	0.8040	0.8889
ecoli-0-2-6-7_vs_3-5	0.7894	0.8236	0.7894	0.8236

As it can be clear, the results are nearly correspondent with the results in the paper and the minor differences are due to the randomness in 5-Fold CV, however, the paper contains an error in defining the *F-Measure* which was implemented correctly in both this code and their own.

Decision Tree

Problem 1

In decision trees, over-fitting occurs when the tree is designed so as to perfectly fit all samples in the training data set. Thus it ends up with branches with strict rules of sparse data. Thus this effects the accuracy when predicting samples that are not part of the training set.

Problem 2

Pruning is a method to deal with over-fitting in decision trees. We can use pre-pruning (stop growing the tree after a certain depth) or post-pruning which is trimming the tree after it is built. The second method is more desirable due to the fact that in a lot of situations we can't derive a good depth before training.

Problem 3

Based on the book **Understanding Machine Learning**:

> A random forest is a classifier consisting of a collection of decision trees, where each tree is constructed by applying an algorithm A on the training set S and an additional random vector, θ, where θ is sampled i.i.d. from some distribution. The prediction of the random forest is obtained by a majority vote over the predictions of the individual trees.
> To specify a particular random forest, we need to define the algorithm A and the distribution over θ. There are many ways to do this and here we describe one particular option. We generate θ as follows. First, we take a random subsample from S with replacements; namely, we sample a new training set S' of size m' using the uniform distribution over S. Second, we construct a sequence $I_1, I_2, \ldots,$, where each I_t is a subset of $[d]$ of size k, which is generated by sampling uniformly at random elements from $[d]$. All these random variables form the vector θ. Then,the algorithm A grows a decision tree (e.g., using the ID3 algorithm) based on the sample S', where at each splitting stage of the algorithm, the algorithm is restricted to choosing a feature that maximizes Gain from the set It. Intuitively, if k is small, this restriction may prevent overfitting.

Here are some advantages of using random forests:

Can perform both classification and regression tasks
Handle the missing values and maintains accuracy for missing data
Won't overfit the model
Handle large data set with higher dimensionality

However random forest does have some draw backs and the most notable one is the little control we have on what the model does and its lack of interpretability.

Problem 4

a.
- Accuracy= 73.68%
- Confusion Matrix:

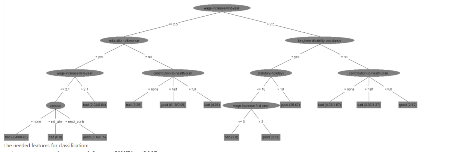
Confusion	Bad	Good
Bad	14	6
Good	9	28

- TN= 14
- FN= 9
- FP= 6
- TP= 28
- Precision= $\frac{TP}{TP+FP}$ = 0.8235
- Recall= $\frac{TP}{TP+FN}$ = 0.7567
- F1= $2 \times \frac{Precision \times Recall}{Precision + Recall}$ = 0.7887
- 

- The needed features for classification:
  1. $wage-increase-first-year = 3$, So we go *RIGHT* from *ROOT*
  2. $statutory-holidays = 12$, So we go *RIGHT* again,
  3. The label is *good*

b.
- The *Pruned* parameter controls whether the tree should be pruned or not, unpruned trees are prone to overfitting the model.
- Accuracy= 78.94%
- Confusion Matrix:

Confusion	Bad	Good
Bad	14	6
Good	6	31

- TN= 14
- FN= 6
- FP= 6
- TP= 31
- Precision= $\frac{TP}{TP+FP}$ = 0.8378
- Recall= $\frac{TP}{TP+FN}$ = 0.8378
- F1= $2 \times \frac{Precision \times Recall}{Precision + Recall}$ = 0.8378
- 

- The needed features for classification:
  1. $wage-increase-first-year = 3$, So we go *RIGHT* from *ROOT*
  2. $longterm-disability-assistance = yes$, So we go *LEFT*,
  3. $statutory-holidays = 12$, So we go *RIGHT*,
  4. The label is *good*
- As can be seen due to the lack of pruning in the algorithm this tree has a lot more leaves and branches than the pruned one.

c.
- Accuracy= 89.47%
- Confusion Matrix:

Confusion	Bad	Good
Bad	16	4
Good	2	35

- TN= 16
- FN= 2
- FP= 4
- TP= 35
- Precision= $\frac{TP}{TP+FP}$ = 0.8974
- Recall= $\frac{TP}{TP+FN}$ = 0.9459
- F1= $2 \times \frac{Precision \times Recall}{Precision + Recall}$ = 0.9210
- The number of trees is 100 and Weka used all features to create the forest

d. As can be seen from the results, Random Forest obtained the best accuracy and in general, due to dealing with the variance part of the error it will achieve better performance.