Project 5 & Project 9 report

Xiaolin Zhang 2019124047 6604080
Ziyang Xu 2019124028 6603452
Wangzhihui Mei 2019124044 6603385
Xiaohuan Pei 2019124022 6603592
CCNU-UOW JI

1 Project 5

1.1 Dataset: Artificial Characters Learning Problem

This database has been artificially generated by using a first order theory which describes the structure of ten capital letters of the English alphabet and a random choice theorem prover which accounts for etherogeneity in the instances. The capital letters represented are the following: A, C, D, E, F, G, H, L, P, R. Each instance is structured and is described by a set of segments (lines) which resemble the way an automatic program would segment an image.

Each instance is stored in a separate file whose format is the following:

```
CLASS OBJNUM TYPE XX1 YY1 XX2 YY2 SIZE   DIAG
1     0      line 0   0   0   13  13.00  45.28
1     1      line 20  0   22  15  15.13  45.28
1     2      line 0   13  22  15  22.09  45.28
1     3      line 0   13  0   27  14.00  45.28
1     4      line 22  15  23  39  24.02  45.28
1     5      line 0   27  23  39  25.94  45.28
```

where CLASS is an integer number indicating the class as described below, OBJNUM is an integer identifier of a segment (starting from 0) in the instance and the remaining columns represent attribute values. [1].

The generated character image is like Figure 1

1.2 Data pre-process

The character described by segments is represented as the vertex pair, we transform them to binary grid (12×8), like Figure 2.

Then the problem is transformed into one like the MNIST classification problem. The feature is the flattened 0-1 pixel, whose dimension is 96×1, the label is "A", "C", "D", "E", "F", "G", "H", "L", "P", "R", which correspond to the numbers 0 through 9.

Figure 1: The generated characters

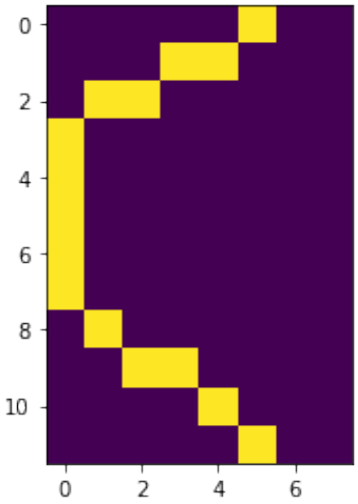

Figure 2: The representation of "C"

We got 6000 training data from the primitive dataset. We shuffled them and use 70% of the data as training data, 30% as test data. The label distribution of training data is like Figure 3

1.3 Ensembling Model

Ensemble modeling is a process where multiple diverse models are created to predict an outcome, either by using many different modeling algorithms or using different training data sets. The ensemble model then aggregates the prediction of each base model and results in once final prediction for the unseen data. The motivation for using ensemble models is to reduce the generalization error of the prediction. As long as the base models are diverse and independent, the prediction error of the model decreases when the ensemble approach is used. The approach seeks the wisdom of crowds in making a prediction. Even though the ensemble model has multiple base models within the model, it acts and performs as a single model. Most

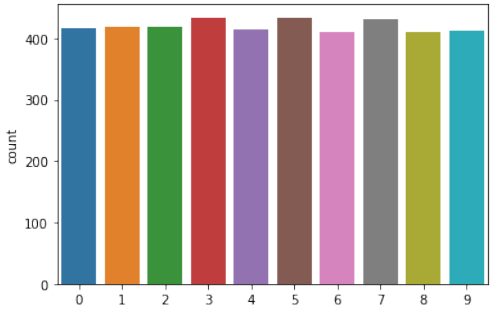

Figure 3: The label distribution of training data

of the practical data mining solutions utilize ensemble modeling techniques. Chapter 4 Classification covers the approaches of different ensemble modeling techniques and their implementation in detail.

In this task, We applied Voting method. Voting is a combination strategy for classification problems in integrated learning. The basic idea is to select the most output class among all machine learning algorithms.

There are two types of output of classification machine learning algorithms: one is to directly output class labels, and the other is to output class probabilities. Using the former to vote is called hard voting (majority/hard voting), and using it to classify is called soft voting (Soft voting). VotingClassifier in sklearn is the implementation of voting method.

In this project, We compared hard voting classifier and soft voting classifier, and got better performance in hard voting classifier, which ensemble Logistic Regression, SGD, SVM, Decision Tree, Random Forest, Extra Tree, MLP as one classifier. The structure is like Figure 4.

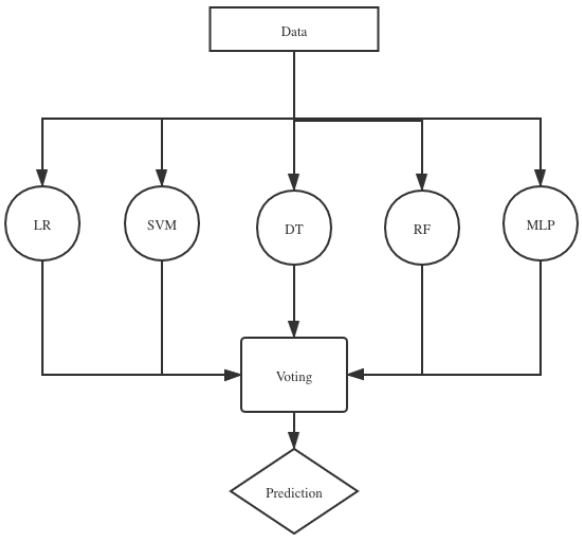

Figure 4: The Ensembled Voting classifier

Performance

We calculated the Recall, Precision, Accuracy and F1-score of the model. As this is a multi-class classification, so each label correspond to one micro value, we can calculate the mean of micro value and get marco value.

Accuracy: 0.97 (+/- 0.00) [Ensemble(hard voting)]
The classification report:

	precision	recall	f1-score	support
A	0.97	0.97	0.97	183
C	0.98	1.00	0.99	182
D	0.98	0.99	0.99	182
E	0.99	0.98	0.98	167
F	0.96	0.93	0.94	185
G	1.00	0.98	0.99	166
H	0.98	0.97	0.98	190
L	1.00	1.00	1.00	168
P	0.93	0.95	0.94	190
R	0.98	0.99	0.99	187
accuracy			0.98	1800
macro avg	0.98	0.98	0.98	1800
weighted avg	0.98	0.98	0.98	1800

Figure 5: The Ensembling (Hard voting) performance

So, we can draw the conclusion that Ensemble Model perform well on both micro and marco f1-score.

1.4 Gradient Boosting Decision Tree

GBDT(Gradient Boosting Decision Tree) is an algorithm that classifies or regresses data by using the additive model (i.e. linear combination of basis functions) and reducing the residual generated in the training process.

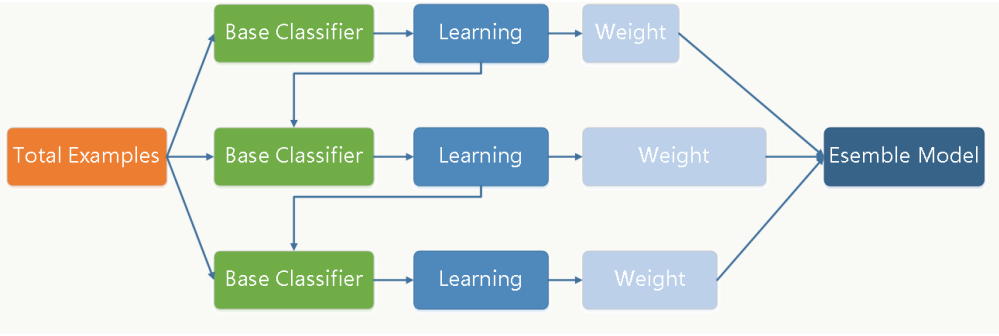

Figure 6: Training process of GBDT

GBDT generates a weak classifier through multiple iterations, each of which is trained on the basis of the residual of the previous one. The requirements for weak classifiers are generally simple enough, with low

variance and high deviation. Because the training process is to continuously improve the accuracy of the final classifier by reducing the deviation (which can be proved here). In general, the weak classifier will choose cart tree. Because of the above high deviation and simple requirements, the depth of each classification regression tree will not be very deep. The final total classifier is the weighted sum of the weak classifiers (that is, the addition model).

Assume that the dataset is represented by $T = \{(x_i, y_i)\}$, our goal is to get estimate $\hat{f(x)}$.The detailed algorithm is as follows:

Algorithm 1: Algorithm of GBDT

Result: $\hat{f(x)}$

Input: dataset $T = \{(x_i, y_i)\}, i = 1, 2, ..., n$, loss function $L(y, f(x))$

Initialize $f_0(x) = argmin_c \sum_{i=1}^{N} L(y_i, x)$

for $m = 1, 2, ..., M$ **do**

  for $n = 1, 2, ..., N$ **do**

    compute $r_{mi} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]$

    for each r_{mi}, fit a CART to get R_{mj} of mth tree

    for each j=1,2,...,J compute $c_{mj} = argmin_c \sum L(y_i, f_{m-1}(x_i) + c)$

    update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj} T(x \in R_{mj})$

  end

  Then compute $\hat{f(x)} = f_M(x) = \sum_{m=1}^{M} \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$

end

Performance

The performance of GBDT is measured by Recall, Precision, Accuracy and F1-score.

```
-------------------------Mean of scores-------------------
 | precision: 0.821429 | recall: 0.824232 | F1: 0.822828 |
 | precision: 0.878469 | recall: 0.885836 | F1: 0.882129 |
 | precision: 0.900679 | recall: 0.905290 | F1: 0.902979 |
-------------------------Std of scores--------------------
 | precision: 0.021844 | recall: 0.018697 | F1: 0.020182 |
 | precision: 0.028888 | recall: 0.028321 | F1: 0.027520 |
 | precision: 0.028124 | recall: 0.027135 | F1: 0.027520 |
```

Figure 7: Performance of GBDT

2 Project 9

2.1 Dataset: Forest Cover Type Prediction

In this project we are asked to predict the forest cover type(the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The actual forest cover type for a given 30 x 30 meter cell was determined from US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from data obtained from the US Geological Survey and USFS. The data is in raw form (not scaled) and contains binary columns of data for qualitative independent variables such as wilderness areas and soil type.

The dataset was given by a csv file, consists of 581012 samples, each sample has 55 fields, with the last one as its label. Then the data was splited to 70% as the training set and 30% as the testing set.

We explored varies classifiers for this project, including Random Forest, Extremely Randomized Trees, Decision Tree, K-Neighbors (KNN), SVM and Artificial Neuron Network (ANN).

2.2 Model construction

The random forest is the first model we choosed to use. The training process lasts nearly 15 minutes. Luckily, the result looks great with accuracy up to 0.957.

```
RandomForest Classifier  0.9567881402606939
```

Figure 8: Random forest results

Besides, we sequence the features influences and draw a diagram of it to better understand the data.

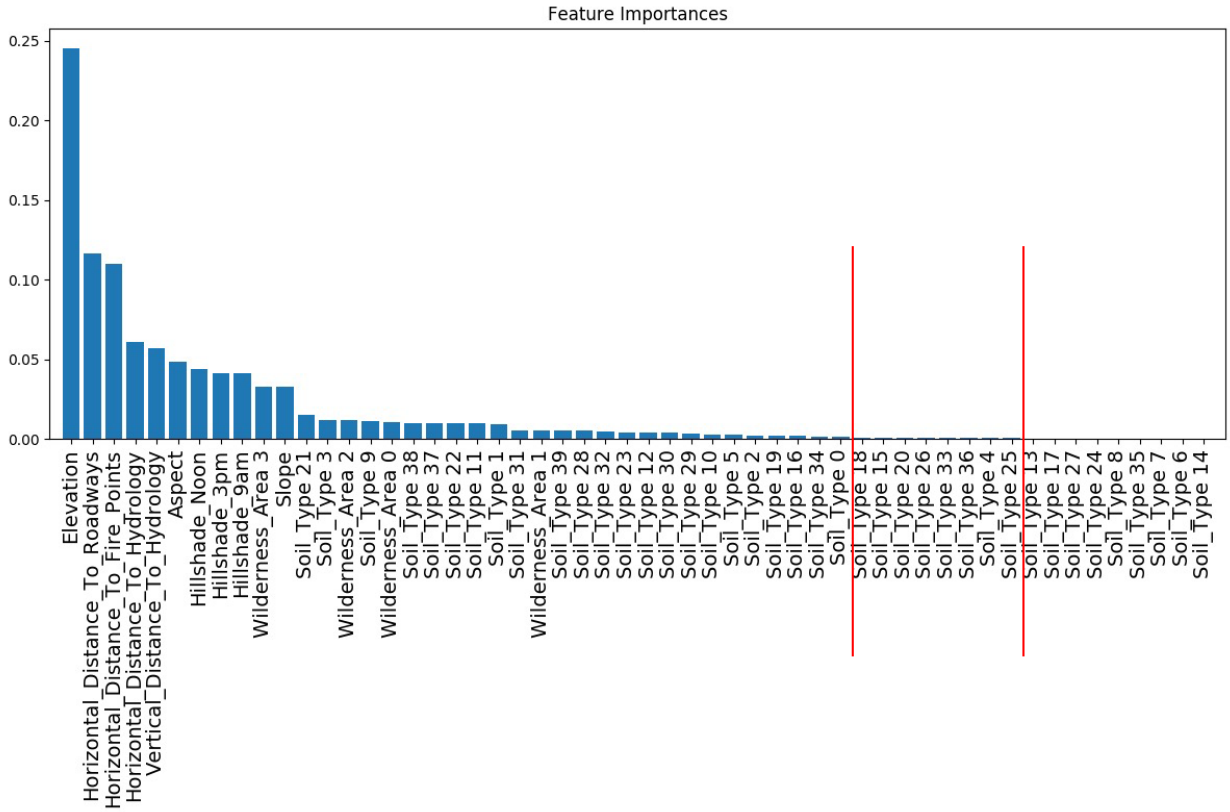

Figure 9: Feature importance

From the diagram, we can see that Evaluation is the most valuable feature with influence up to 0.25. Paying more attention to this feature when training model may enhance the prediction rate. On the countrary, the last 9 features had no influences on the final classificasion result while the last 10th to 17th features had little influences. Therefore we could consider about deleting these 17 features from the datasets to optimal the model construction and save training time. This is the final data after feature selection. The featues have been extracted from 54 to 37. Then the scatters of top-3 features is drawed in Figure 10, we find that these features show great correlation and continuity with Cover_type, which demonstrate their significance in classification.

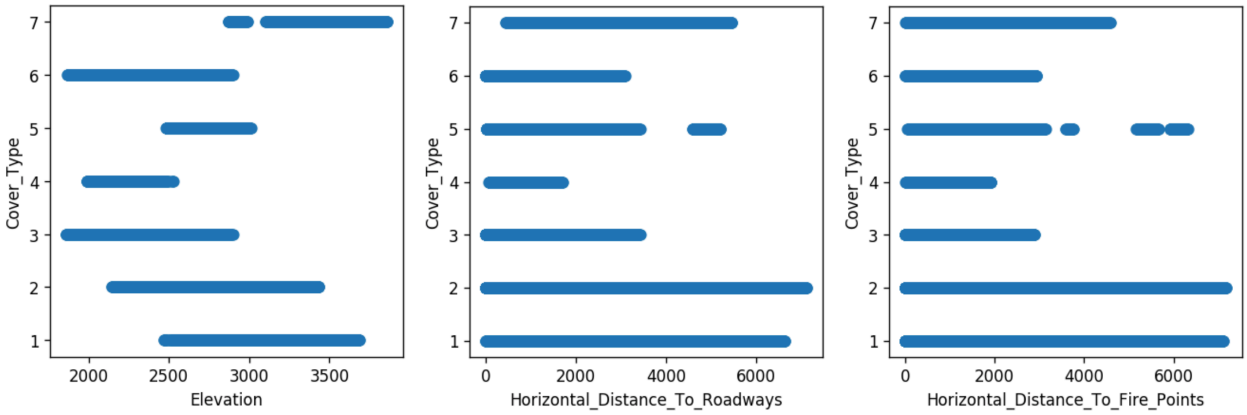

Figure 10: Feature scatters

Then we thought about using the decision tree and knn models. With the help of sklearn library, these can be implemented conveniently. The accuracy for decision tree is 0.9336 while the knn's accuracy is 0.9666, which is slightly higher than decision tree. The result is shown in Figure 11.

```
DecisionTree Classifier  0.9336446667890582
KNeighbors Classifier    0.9665756379658528
```

Figure 11: Decision trees and knn results

The SVM modeled is checked later, while Xiaolin Zhang uses Libsvm and Ziyang Xu uses sklearn. For Libsvm, a python script is developed to convert the original data to the libsvm's format. Libsvm provide a useful tool 'easy.py' in the sub folder "tools" for automatically process the data and search the best parameters. The best c and g its found is 32768 and 0.000122. Using this parameters, we get a accuracy of 86.0929% in Figure 12. The sklearn is also used for SVM, but instead we use the GridSearchCV to search the best

```
optimization finished, #iter = 1706
nu = 0.000013
obj = -178.791978, rho = 0.528543
nSV = 767, nBSV = 0
Total nSV = 37763
Accuracy = 86.0929% (15006/17430) (classification)
```

Figure 12: Libsvm results

parameters C. There are 4 C candidates and 3 fold crossvalidation is used, thus total 12 model is tested, takes about half an hour. Figure 13 shows that 100 is the best. Using the c=100 to train the svm, we get the accuracy of 99.98%.

Then, we used tendorflow as toolbox to consrtruct and train an artificial neuron network as classifier. Considering of the gigantic features size (54), we decided to build a deep learning network with more than 2 hidden layers to acquire better performance. After several combinations and tries, we found a structure showing relatively superior performance, which has 4 hidden layers with 50 , 80 , 80 neurons respectively

```
In [47]: classifier = GridSearchCV(svm.SVC(),svm_para,cv=3,verbose=2)
         classifier.fit(x_data,y_data)
         classifier.best_params_
         classifier.cv_results_

Fitting 3 folds for each of 4 candidates, totalling 12 fits
[CV] C=1, kernel=rbf .................................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] ................................... C=1, kernel=rbf, total= 4.2min
[CV] C=1, kernel=rbf .................................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  4.2min remaining:    0.0s
[CV] ................................... C=1, kernel=rbf, total= 4.0min
[CV] C=1, kernel=rbf .................................................
[CV] ................................... C=1, kernel=rbf, total= 4.1min
[CV] C=10, kernel=rbf ................................................
[CV] .................................. C=10, kernel=rbf, total= 2.2min
[CV] C=10, kernel=rbf ................................................
[CV] .................................. C=10, kernel=rbf, total= 2.4min
[CV] C=10, kernel=rbf ................................................
[CV] .................................. C=10, kernel=rbf, total= 2.6min
[CV] C=100, kernel=rbf ...............................................

In [48]: classifier.best_params_
Out[48]: {'C': 100, 'kernel': 'rbf'}
```

Figure 13: Gridsearch

on each layer and relu as activation function. Besides, a dropout layer is introduced to randomly inactivate neurons to avoid overfitting. The training process lasted for nearly 5 minutes and the testing accuracy reached to 0.7973. To better observe the training process, pyplot is used to draw diagrams in Figure 14 of accuracy and loss trends. The loss remains still as the training goes, thus we believe it has converged.

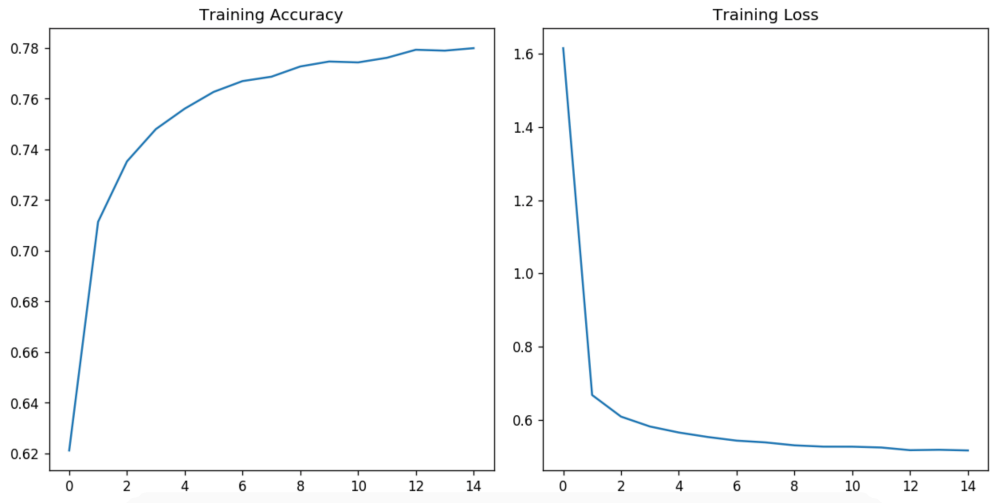

Figure 14: Train process

Finally, Extremely Randomized Trees Classifier(Extra Trees Classifier) is tested. It is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a forest to output its classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest. Gridsearch is also used to identify the best parameters. It turns out that max_depth : 10, $max\_features$: 0.3 and $n\_estimators$: 100 is a

Model	Accuracy
Random Forest	0.9568
Decision Tree	0.9336
KNN	0.9666
SVM	0.9998
ANN	0.7973
ExtraTree	1.0

Table 1: Caption

good choice. Then we build a ExtraTree based on these parameter, get the 100% accuracy on testset. The result is shown in Figure 15

```
In [73]: print ('Best accuracy obtained: {}'.format(ETC.best_score_))
         print ('Parameters:')
         for key, value in ETC.best_params_.items():
             print('\t{}:{}'.format(key,value))

         Best accuracy obtained: 1.0
         Parameters:
                 max_depth:10
                 max_features:0.3
                 n_estimators:100

In [76]: etc = ExtraTreesClassifier(bootstrap=True, oob_score=True, n_estimators=100, max_depth=10, max_features=0.3, \
                                    min_samples_leaf=1)

         etc.fit(X_train, y_train)
         yy_pred = etc.predict(X_test_temp)

In [78]: accuracy = len([yy_pred[i] for i in range(0, len(yy_pred)) if yy_pred[i] == y_test[i]]) / len(yy_pred)
         accuracy

Out[78]: 1.0
```

Figure 15: ExtraTree results

All the classifier used in project 9 along with their results is shown in the Table 1:

References

[1] Schapire, R. E. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336.