Final Project

Name: Yuan Li, Qiang Luo
netID: yl6606, ql967
Email: yl6606@nyu.edu, ql967@nyu.edu

Extension 1: Decision Tree and Random Forest

1. Inspiration

Generally, we consider Decision Tree as a kind of weak classifier. If we can combine some weak classifiers, is it possible that we can get better predictions? Actually, we can. We will use Hoeffding inequality to prove this idea.

Considering a binary classification problem, we use M individual classifiers g_i which error probabilities are σ and they are independent of each other. We use simple voting for ensemble learning, that is, the classification result is the label with the largest amount in the independent classifiers:

$$G(x) = sign(\sum_{i=1}^{M} g_i(x))$$

Set the true results are $f(x)$. Based on Hoeffding inequality, the error rate after ensemble learning should be:

$$P(G(x) \neq f(x)) \leq exp(-\frac{1}{2}M(1-2\sigma)^2)$$

When the number of independent base learners M becomes larger, the error probability becomes smaller, even would be close to 0, which is also in line with the intuitive idea: the probability that most people make errors at the same time is relatively low.

There are two general algorithms for ensemble learning: **Random Forest/Bagging**, **Boosting**. We choose to use Random Forest.

2. Experiment and Result

We tried two datasets – digits data in sklearn and Heart Disease UCI data from UCI. Here are the accuracies.

	DT*	RF**(sklearn)	RF(Our)
digits	85.111%	96.222%	93.333%
Heart Diesease	64.474%	68.421%	61.842%

* DT means Decision Tree.
** RF means Random Forest.

The accuracies show that Random Forest is better than Decision Tree on both dataset. Comparing the results of Random Forest written by us and sklean, we got a lower accuracy than sklean's version. We will explain these results in the next section.

3. Explanation of results

We can find that the accuracy of Random Forest is higher than the result of Decision Tree, which meets our previous assumption. However, the accuracy of our implementation is lower than sklean's version. After discussion, we think it is because of the preprossing method for continuous variables.

The two datasets' variables are both numerical, so in order to meet the needs of our algorithms' input, we need to prepross these variables. Out method is to split the variables into 3 preiods based on their values. However, this will lose some information in the variables. For example, we may put two values into the same period, but when the value of this variable is separately equal to both, their prediction results are different.

Actually, sklearn uses a more complex preprocessing method to handle continuous variables. This may also explain three algorithms all receive low accuracy on Heart Diesease data.

Extension 2: Neural Network's new optimizers

1. Inspiration

The optimizer of Neural Network in assignment 8 is gradient descent method, each parameter is reduced according to the direction of the gradient in pursuit of minimizing the loss function. Actually, there are two new methods to update parameters in Neural Network, called momentum method and adaptive methods.

The momentum method aims to change the gradient of the current position parameter by the gradient of each parameter in the previous iteration. It can accelerate the update speed where the gradient is stable, and stabilize the gradient where the gradient is unstable. The basic momentum parameter update expression is as below:

$$v = \alpha * v - lr * \Delta W$$
$$W = W + v$$

* α is momentum coefficient.
** v is momentum.

We tried another completely different idea – adaptive method. It dynamically updates the learning rate of each parameter through the historical gradient of each parameter, so that the update rate of each parameter can be gradually reduced. if the previous gradient increases faster, the learning rate decreases faster. Otherwise, the learning rate decreases more slowly. The update expression is as below:

$$h = h + \Delta W * \Delta W$$
$$W = W - lr * \frac{1}{\delta + \sqrt{h}} * \Delta W$$

* δ is to avoid that the divisor is 0.

This method is also called AdaGrad. Here's an improvement of AdaGrad, called RMSprop. The problem with AdaGrad is that the learning rate will continue to decline. This will cause many tasks to reduce the learning rate excessively before reaching the optimal solution, so RMSprop uses an exponential decay average to slowly discard the previous gradient history. This operation prevents the learning rate from decreasing prematurely.

The new update expression is as below:

$$h = \rho * h + (1 - \rho) * \Delta W * \Delta W$$
$$W = W - lr * \frac{1}{\delta + \sqrt{h}} * \Delta W$$

* δ is to avoid that the divisor is 0.
** ρ is decay rate.

2. Experiment and Result

We tried two datasets – digits data in sklearn and Bank Marketing data from UCI. Here are the accuracies.

	SGD*	RMSprop (Keras)	RMSprop (Our)	AdaGrad (Keras)	AdaGrad (Our)
digits	85.556%	95.111%	96.0%	93.111%	96.889%
Bank Marketing	90.007%	91.201%	90.968%	91.658%	91.541%

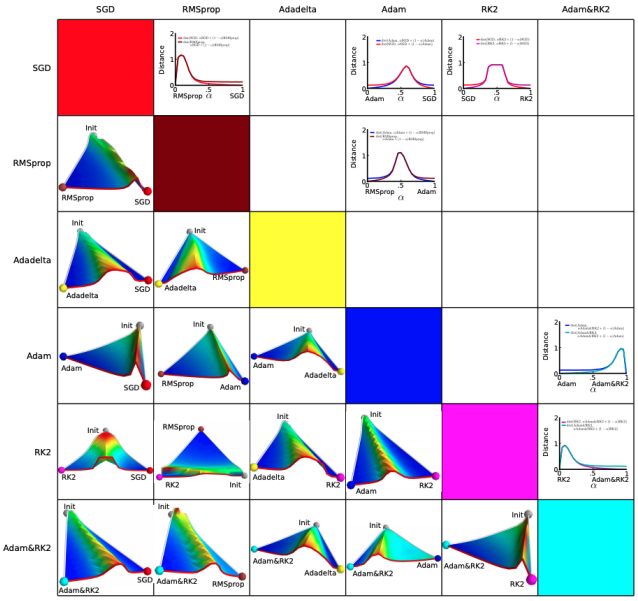
* We changed the original optimizer from GD to SGD, in order to train more quickly.

Firstly, we used package – Keras to build the extension Neural Network, then we implemented our own extensions based on code from assignment 8. We found that the two optimizers all got better accuracy than SGD. On digits data, the accuracy was even 10% better than SGD. We will explain these results in the next section.

3. Explanation of results

The reason they get different accuracy is because the gradient direction of the descent is different. Due to the different descent directions, different algorithms may lead to completely different local optimums.

"An empirical analysis of the optimization of deep network loss surfaces" made an interesting experiment. They map the hyperplane formed by the objective function value and corresponding parameters to a three-dimensional space, so that we can visually see how each algorithm finds the lowest point on the hyperplane. Here is the result.



The horizontal and vertical coordinates represent the feature space after dimensionality reduction, and the area color represents the change in the value of the objective function. Red is the plateau and blue is the depression. What they did was a paired experiment, where the two algorithms started from the same initialization location, and then compared the results of the optimization.

It can be seen that almost any two algorithms have reached different depressions, and a high plateau is often separated between them. This shows that when different algorithms are on the plateau, they choose different descent directions.

Our experiment shows that AdaGrad and RMSprop are better than SGD. However, some other people think the opposite. *"The Marginal Value of Adaptive Gradient Methods in Machine Learning"* made an experiment, which proved that AdaGrad and RMSprop might be worse than original SGD. Due to limited time, we did not reproduce their experimental results, but this is a question worth thinking

about.

Extension 3: Neural Network's new activation

1. Inspiration

Considering a multi-label classification problem, we can treat the output of the last layer as the probability of different classes, and then select the class corresponding to the maximum value. However, taking the maximum can't be derivative. For the Neural Network in assignment 8, it still uses derivative of sigmoid to update weights.

Actually, here's a method to handle this problem. We can use the exponent of the output layer, then take the maximum. Because exponent can be derivative, it also solved how to update weights in back propagation. This method is called softmax. The expression is as below:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}$$

2. Experiment and Result

We tried two datasets – digits data in sklearn and Bank Marketing data from UCI. Here are the accuracies.

	sigmoid	softmax (Keras)	softmax (Our)
digits	85.556%	94.667%	93.556%
Bank Marketing	90.007%	91.192%	91.308%

* We changed the original optimizer from GD to SGD, in order to train more quickly.

Firstly, we used package – Keras to build the extension Neural Network, then we implemented our own extensions based on code from assignment 8. We will explain these results in the next section.

3. Explanation of results

On digits data, softmax made an obviously improvement of accuracy. However, on Bank Marketing data, it didn't. The difference between two data is reasonable. Digits data has 10 classes, while Bank Marketing data has only 2 classes, so softmax did not make much improvement on it.

In order to make the softmax function more stable at the level of numerical calculation and avoid nan in its output, we performed one more operation on the input vector – multiplied a constant C on the numerator and denominator. The expression is a little different from the original softmax function.

$$softmax(x_i) = \frac{e^{x_i + log(C)}}{\sum_{k=1}^{N} e^{x_k + log(C)}}$$

We usually choose $-max(x_i)$ as C.

References

1. https://keras.io/optimizers/
2. https://www.itcodemonkey.com/article/6725.html
3. *The Marginal Value of Adaptive Gradient Methods in Machine Learning* https://arxiv.org/abs/1705.08292
4. *An empirical analysis of the optimization of deep network loss surfaces* https://arxiv.org/abs/1612.04010
5. https://www.jianshu.com/p/3c8e22adf737