School of Computing and Information Systems
The University of Melbourne
COMP90049 Introduction to Machine Learning (Semester 1, 2023)

Week 11: Solution

1. What is **gradient descent**? Why is it important?

   Gradient descent is an iterative optimization algorithm used for minimizing the cost function or error function in machine learning and other optimization problems. The algorithm works by iteratively adjusting the model's parameters in the opposite direction of the gradient of the cost function, in order to minimize the error or cost.

   Gradient descent is particularly useful in situations where there is no closed-form solution available for finding the optimal parameters, as it can iteratively update the parameters until convergence. This makes it a popular choice for machine learning tasks such as linear and logistic regression, neural networks, and deep learning, where the goal is to find the weights that minimize the error or cost function over a training dataset.

2. How is **Logistic Regression** similar to **Naive Baye***s* and how is it different? In what circumstances would the former be preferable, and in what circumstances would the latter?

   Similarity:
   Both methods are classification methods, attempting to predict the most suitable class Y for a test instance X. Both methods compute P(Y|X) and predict the class with the highest probability. Thus, both methods are probabilistic machine learning models, basing their predictions around probabilities. Both classifiers have parameters, which are maximized using the Maximum Likelihood Principle. Both classifiers require input instances to be represented through a pre-defined set of features (and their appropriate values).

   Difference:
   In Naïve Bayes we convert the learning problem using Bayes rule which models P(x,y) by using likelihoods p(x|y) and priors p(y), and maximizes the joint likelihood of the training data to find the best parameters. Naïve Bayes includes the *conditional feature independence* assumption, which ensures that the likelihood p(x|y) can be effectively estimated. But Logistic Regression models P(y|x) *directly* and maximizes the conditional likelihood of the training data to find the best parameters. By *directly modelling p(y|x), there is no need to estimate p(x|y)*, so Logistic Regression does not require the conditional feature independence assumption.

   Moreover, Naïve Bayes is a *generative model*. Let's consider a visual metaphor: imagine we're trying to distinguish dog images from cat images. A generative model would have the goal of understanding what dogs look like and what cats look like. But Logistic Regression is a *discriminative* model. It means Logistic Regression is only trying to learn to *distinguish* between the classes (without learning much about them). So, in our example, if all the dogs in the training data are wearing collars and the cats aren't. That one feature would neatly separate the classes and the model only predict the label for the test instances using that one feature (by assigning a very high weight to that feature). If you ask such a model what it knows about cats *all it can* say is that they don't wear collars!

   Preferability:
   Logistic Regression generally tends to outperform Naïve Bayes, as it doesn't require conditional independence assumption. Naïve Bayes is conceptually simpler, and easier to implement, its parameters can be estimated in closed form, whereas for Logistic Regression we have to adopt an iterative optimization method which can be time-consuming.

3. Describe how to build Random Forrest for a given data?

Assume the training data set contains N instances and M attributes. Random Forest is an ensemble learning method that constructs multiple decision trees during training and output the class using majority voting following the paths in individual trees. Here is a high-level description of how to build a Random Forest for a given data:

- Feature selection: For each tree in the forest, randomly select a subset (k) of features (M) to use for training. This helps to reduce overfitting and improve the generalization performance of the model.

- Decision tree construction: Using the selected features, construct a decision tree by randomly selecting N training instances with replacement similar to bagging.

- Random forest construction: Combine the decision trees into a random forest by taking the majority vote of the individual trees.

(a) What is the benefit of bagging?

Bagging helps to build uncorrelated decision trees.

(b) What is the impact of the size of the number of trees in Random Forest?

Larger number of trees (assuming the trees are uncorrelated) reduce the variance of the Random Tree. However, larger random forest takes more time to train and classify. The complexity grows linearly with the number of trees.

(c) What will happen if the random number features chosen for splitting nodes in a Random Forest is very large?

As the random number features chosen increases the correlation between the trees increases and the accuracy of Random Forrest goes down.

4. Under what circumstances we prefer stacking to boosting and bagging?

Bagging can reduce overfitting by reducing variance, as it generates multiple models by training on different subsets of data and aggregating their predictions. However, if the base models used in bagging are too complex, they may overfit on their respective subsets of data, leading to an overall overfitting of the bagging ensemble.

Boosting, on the other hand, can reduce bias by iteratively adjusting the weights of misclassified data points, which can improve the fit of the model to the data. However, if the base models are too simple, boosting can overfit by continuously fitting the misclassified points of the previous models.

Stacking, as a meta-learning method, combines the strengths of both bagging and boosting while mitigating their weaknesses. It can reduce both bias and variance by stacking models trained with different algorithms. However, it requires more computational resources and can be more complex to implement and tune than bagging or boosting. Therefore, stacking may be preferred over bagging or boosting when the data is complex, and we want to consider predictions from heterogeneous models.

5. Let's revisit the logic behind the voting method of classifier combination (used in Bagging, Random Forests, and Boosting to some extent). We are assuming that *the errors between all classifiers are uncorrelated*

(a) First, let's assume our three independent classifiers all have the error rate of e = 0.4, calculated over 1000 instances with binary labels (500 A and 500 B).

(i) Build the confusion matrices for these classifiers, based on the assumptions above.

All our systems have the error rate of 0.4. It means that in 40% of the times our system makes a mistake and in 60% of the time it makes a correct prediction. So, from 500 instances with label A, 300 instances will be (correctly) predicted with the label A, and 200 instances will be labelled (incorrectly) as B.

Now since we are doing an ensembled method, we can assume that from the 300 instances that system 1 labelled as A (and 200 instances labelled as B), again 60% will be labelled (correctly) as A and 40% (incorrectly) as B, and so on.

Table below contains all the counts for our three-classifier ensemble.

Actual class	#	P1	# for Sys1	P2	# for Sys 2	P3	# for Sys 3
A	500	A	(500 x 0.6=) 300	A	(300 x 0.6=) 180	A	(180*0.6=) 108
						B	(180*0.4=) 72
				B	(300 x 0.4=) 120	A	(120*0.6=) 72
						B	(120*0.4=) 48
		B	(500 x 0.4=) 200	A	(200 x 0.6=) 120	A	(120*0.6=) 72
						B	(120*0.4=) 48
				B	(200 x 0.4=) 80	A	(80*0.6=) 48
						B	(80*0.4=) 32
B	500	A	(500 x 0.4=) 200	A	(200 x 0.4=) 80	A	(80*0.4=) 32
						B	(80*0.6=) 48
				B	(200 x 0.6=) 120	A	(120*0.4=) 48
						B	(120*0.6=) 72
		B	(500 x 0.6=) 300	A	(300 x 0.4=) 120	A	(120*0.4=) 48
						B	(120*0.6=) 72
				B	(300 x 0.6=) 180	A	(180*0.4=) 72
						B	(180*0.6=) 108

(ii) Using that the majority voting, what the expected error rate of the voting ensemble?

Since we have 3 systems, using a majority voting is very easy. For all the predicted labels we check the results of the 3 system and if 2 out of 3 votes for one class we chose that class as the label for the whole system. The results for the voting system are demonstrated in the following table, where the incorrect predictions are highlighted.

Actual class	#	Pred1	#	Pred2	#	Pred3	#	Majority Vote
A	500	A	300	A	180	A	108	Maj(A,A,A)= A
						B	72	Maj(A,A,B)= A
				B	120	A	72	Maj(A,B,A)= A
						B	48	Maj(A,B,B)= B
		B	200	A	120	A	72	Maj(B,A,A)= A
						B	48	Maj(B,A,B)= B
				B	80	A	48	Maj(B,B,A)= B
						B	32	Maj(B,B,B)= B
B	500	A	200	A	80	A	32	Maj(A,A,A)= A
						B	48	Maj(A,A,B)= A

				B	120	A	48	Maj(A,B,A)= A
						B	72	Maj(A,B,B)= B
		B	300	A	120	A	48	Maj(B,A,A)= A
						B	72	Maj(B,A,B)= B
				B	180	A	72	Maj(B,B,A)= B
						B	108	Maj(B,B,B)= B

From this table we can identify that the total count of incorrectly classified instances is:

error = 48 + 48 + 48 + 32 + 32 + 48 + 48 + 48 = 352

Therefore, the error rate of the final system (the ensemble of three learners with error rate of 0.4) is $\frac{352}{1000} = 35.2\% = 0.352$. It is better that the error rate of each system individually. It is mostly because using three system has allowed us to disambiguate the instances where each system cannot classify correctly. In other words, the voting system helps the learners to correct each other's mistake.

This relies on the assumption of errors being uncorrelated: if the errors were perfectly correlated, we would see no improvement; if the errors were mostly correlated, we would see only a little improvement.

(b) Now consider three classifiers, first with $e_1 = 0.1$, the second and third with $e_2 = e_3 = 0.2$.

(i) Build the confusion matrices.

Similar to part A we can build a combined confusion matrix for all three systems, where the first system is 90% makes correct prediction (and therefore makes incorrect prediction for 10% of the instances). And the two next systems make correct predictions only for 80% of the instances (and so each system makes 20% incorrect predictions).The following table contains all the counts for the all different combination of the systems in this ensemble.

A	500	A	(500 x 0.9=) 450	A	(450 x 0.8=) 360	A	288
						B	72
				B	(450 x 0.2=) 90	A	81
						B	18
		B	(500 x 0.1=) 50	A	(50x 0.8=) 40	A	36
						B	8
				B	(50 x 0.2=) 10	A	8
						B	2
B	500	A	(500 x 0.1=) 50	A	(50 x 0.2=) 10	A	2
						B	8
				B	(50 x 0.8=) 40	A	8
						B	32
		B	(500 x 0.9=) 450	A	(450 x 0.2=) 90	A	18
						B	72
				B	(450 x 0.8=) 360	A	72
						B	288

(ii) Using the majority voting, what the expected error rate of the voting ensemble?

The results for the voting system are demonstrated in the following table, where the incorrect predictions are highlighted.

Actual class	#	Pred1	#	Pred2	#	Pred3	#	Majority Vote
A	500	A	450	A	360	A	288	Maj(A,A,A)= A
						B	72	Maj(A,A,B)= A
				B	90	A	81	Maj(A,B,A)= A
						B	18	Maj(A,B,B)= B
		B	50	A	40	A	36	Maj(B,A,A)= A
						B	8	Maj(B,A,B)= B
				B	10	A	8	Maj(B,B,A)= B
						B	2	Maj(B,B,B)= B
B	500	A	50	A	10	A	2	Maj(A,A,A)= A
						B	8	Maj(A,A,B)= A
				B	40	A	8	Maj(A,B,A)= A
						B	32	Maj(A,B,B)= B
		B	450	A	90	A	18	Maj(B,A,A)= A
						B	72	Maj(B,A,B)= B
				B	360	A	72	Maj(B,B,A)= B
						B	288	Maj(B,B,B)= B

From this table we can identify that the total count of incorrectly classified instances is:

Error = 18 + 8 + 8 + 2 + 2 + 8 + 8 + 18 = 72

Therefore, the error rate of the final system (the ensemble of three learners $e_1 = 0.1$ and $e_2 = e_3 = 0.2$) is now $\frac{72}{1000} = 7.2\% = 0.072$. It is better that the error rate of the best system.

(iii) What if we relax our assumption of independent errors? In other words, what will happen if the errors between the systems were very highly correlated instead? (Systems make similar mistakes.)

Basically, if all the systems make the same predictions; the error rate will be roughly the same as the correlated classifiers, and voting is unlikely to improve the ensemble. Even if two of the classifiers are correlated, and the third is uncorrelated, the two correlated systems will tend to “out-vote” the third system on erroneous instances.

Therefore, if we want to use ensemble method, it’s best to use uncorrelated learners (classifiers).