PROJECT-3
Classification algorithms

Abhinvav Kumar(akumar39)

Sachin Kumar Kuppayya(skuppayy)

Vinooth Rao Kulkarni (vinoothr)

k-nearest neighbor

Algorithm:

- For each testing example that needs to be labelled calculate the distance between the testing example and each training examples**. Distance calculation** is done based on the continuous and categorical features as mentioned in the below sections.
- **Sort the distances** and select the k-nearest neighbor for the testing example in hand. We select the K neighbor based on the accuracy. Final value of K is selected which gives the maximum accuracy.
- Since we have binary classification for labelling, obtain the count of examples with each label out of the K-nearest neighbors obtained in the previous step. The class label for the test example is the **maximum** label obtained out of the k-nearest neighbors.

Choice of K:

As a rule of thumb and since the data given is for binary classification we iterate for odd values of k = 3, 5, as even value of k may divide the example data into half for each label. For each value of k find expected accuracy for the given data set based on the application requirement. We observed that small value of k will affect the classification and very large value will not linearly increase the accuracy.

Categorical and Continuous features:

For continuous features we have used Euclidian distance directly as the distance metric. For categorical features we map the features to 0,1,2,3 ….to the features with categorical values. Then Euclidian distance is calculated as before for the categorical features. This helps to combine the distance of the continuous features and categorical features.

Results:

Dataset1

	Accuracy	Precision	Recall	F1-Measure
k = 3	0.961341	0.96475	0.952515	0.956526038
k = 5	0.966635	0.968724	0.958674	0.962068871
k = 7	0.963127	0.966079	0.95443	0.958109735
k = 9	0.966635	0.970596	0.958753	0.962800462

	Dataset2			
	Accuracy	Precision	Recall	F1-Measure
k = 3	0.66901	0.63736	0.615737	0.61127546
k = 5	0.658187	0.615252	0.591947	0.585306025
k = 7	0.657956	0.605821	0.592966	0.585940665
k = 9	0.690426	0.65629	0.618805	0.611202783

Result Analysis:

- We observe that accuracy is very high for Dataset1
- Computation of the accuracy is quick as it does not require preprocessing and is based on lazy learning
- We observe that algorithm is more suitable when the data has all continuous attributes

Cross Validation:

We have performed the cross validation in 10-fold on the Knn algorithm.

We first split the original data as the test data and train data for 10-fold validation and observer the resulting performance metrics.

For the next fold next set of test data and training data are obtained from the original data. The above procedure is repeated for 10 times to cover the entire data set.

Average accuracy is calculated from the above iterations.

Accuracy Results for cross validation:

Dataset1	Dataset2
0.964912	0.574468
0.964912	0.659574
0.964912	0.673913
0.964912	0.695652
0.912281	0.608696
0.982456	0.586957
1	0.717391
0.982456	0.782609
0.947368	0.565217
0.982143	0.717391

Pros:

- simple and instance-based learning algorithm
- Lazy learning and skips training phase
- Useful for nonlinear data

Cons:

- Testing phase is computationally costlier in terms of both time and memory because of the calculation of the distance required between each test data and training data. Sorting of the distance to choose k-nearest neighbors for every test data is time consuming.
- Accuracy is reduced when the class distribution is skewed because of the majority voting. Examples of more frequent classes will dominate the prediction for the new examples because
  They tend to be common among k-nearest neighbors.
- Small values of K are sensitive to noise and large value of k may include points from other class.

Decision Tree

Algorithm:

- Build the decision tree based on the training data by splitting the data according to the Gini index calculation for impurity. Correspondingly add the true and false data to the left and right nodes.
- Gene index calculates the impurity at each column and each split in the training data.
- Repeat the above steps recursively for the left and right nodes until all the records have the same class label or the maximum depth required is reached.
- After the maximum depth, for the remaining data most common class label is assigned. Large depths are avoided to reduce the irregular patterns.

Categorical and Continuous features:

We identify and mark the continuous and categorical features in the data set.

For the categorical features / columns we identify all the **unique values for that column**. These values are **mapped arbitrarily to 0,1, 2,** This is required in order to calculate the Gene index as it considers all the possible combinations of values for column it is applied for splitting the training data. For ex: Sunny, cold and humid map to 0,1,2 and combinations of [0,1], [1,2], [0,2] etc. are considered instead of the original features.

For the continuous columns / features Gini index calculates the split at a row where the impurity for that feature is minimum and the values for the **feature > split value** are put on right tree and values for the **feature < split value** are put on the left tree. This is done recursively over all the training data and required **depth**.

Best Feature:

As mentioned above, the Gini index is calculated after identifying whether the feature is categorical or continuous. We select the split value which corresponds the **least impurity / Gini value.** This process is repeated for each node recursively so that we always choose the best feature at that **node and split the training data** base on the split value accordingly.

Post Processing:

We analyze the results for the test data by cross validation.

Results:

	dataset1	dataset2
Accuracy	0.929543	0.651388
Precision	0.931822	0.548968
Recall	0.931822	0.519447
F1measure	0.921419	0.469644

Result Analysis:

- Accuracy of the decision tree is lower than the KNN
- We observed that the algorithms perform slow for dataset1 as it has more features compared to dataset2

Cross Validation:

We have performed the cross validation in 10-fold on the Knn algorithm.

We first split the original data as the test data and train data for 10-fold validation and observer the resulting performance metrics.

For the next fold next set of test data and training data are obtained from the original data. The above procedure is repeated for 10 times to cover the entire data set.

Average accuracy is calculated from the above iterations.

Accuracy Results for cross validation:

Dataset 1	Dataset2
0.92982456	0.617021
0.94736842	0.744681
0.94736842	0.5
0.98245614	0.695652
0.85964912	0.586957
0.94736842	0.586957
0.92982456	0.73913
0.98245614	0.76087
0.92982456	0.652174
0.83928571	0.630435

Pros:

- Performance is faster for test data as decision tree is built based on the training data
- Easy to interpret the data and analyze the splits at each node and feature
- Accuracy is comparable to other classification algorithms because of the Gini index / info gain calculation considers every combination over all the features

Cons:

- Building initial tree recursively is slow. Each node computes Gene index considering all the combinations of feature values.
- Complexity increases rapidly as the number of features and unique values for each feature increases.
- Changes in Training data will generate and new decision tree which is costly.
- Necessary to post prune to improve the classification accuracy.

Naïve Bayes

Algorithm:

- We separately handle continuous and categorical features. For continuous feature data we assume that data is distributed according to the **Gaussian distribution** for each class.
- We segment the training data by class and compute the **mean and variance** for each **continuous attribute** / features in each class.
- Compute the Probability density using the formula each continuous attribute.

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Compute Prior and Posterior probabilities for Categorical attributes for the following formula used for classification.

$$P(H_i \mid X) = \frac{P(H_i)\,P(X \mid H_i)}{P(X)}$$

Where

P(H|X) is the Class Posterior Probability

P(H) is the Class Prior Probability

P(X|H) Posterior Probability of X conditioned H

P(X) is the prior probability of existence of tuple x.

And H be the Hypothesis that tuple X belong to a class C.

- We observe that P(X) is constant for all the classes which will not affect our classification.

Continuous attribute:

In the firsts step we separately identify the continuous attributes. We calculate the mean and standard deviation for each column corresponding to the class label. Hence for the test data we calculate the probability density for each column of that tuple using the already calculated mean and standard deviation for that column.

Zero Probability:

Naïve Bayes gives zero probability if a tuple does not have particular attribute value. This is fixed by using Laplacian Correction.

Results:

	dataset1	dataset2
Accuracy	0.9349	0.688344
Precision	0.928686	0.673759
Recall	0.93002	0.689375
F1measure	0.928525	0.669798

Result Analysis:

- We observe that Naïve Bayes outperforms all the other algorithms in performance for dataset2
- The above observation is due to the independent attributes in the data set2

Cross Validation:

We have performed the cross validation in 10-fold on the Knn algorithm.

We first split the original data as the test data and train data for 10-fold validation and observer the resulting performance metrics.

For the next fold next set of test data and training data are obtained from the original data. The above procedure is repeated for 10 times to cover the entire data set.

Average accuracy is calculated from the above iterations.

Accuracy Results for cross validation:

Dataset1	Dataset2
0.947368	0.659574
0.929825	0.702128
0.964912	0.782609
0.912281	0.695652
0.894737	0.695652
0.964912	0.608696
0.964912	0.782609
0.964912	0.73913
0.912281	0.586957
0.892857	0.630435

Pros:

- Highly scalable
- Use simple technique for classification
- Efficient for many practical applications
- Requires small number of training data as a result suits many applications
- Converges very fast if the attributes hold no relation

Cons:

- Not suitable for correlated attributed data
- Zero probability issue for categorical data
- Low Accuracy for correlated features

Random Forest

Algorithm:

- Use build decision tree from the previous decision tree algorithm based on **bagging** technique. Bagging is used to generate 'm' training data sets of size 'n' uniformly with replacement to create 'd' decision trees.
- For each of the decision trees in the above steps we **select a subset of 0.2 * number of features** randomly.
- Gene index calculates the impurity at each column and each split in the training data.
- Repeat the above steps recursively for the left and right nodes until all the records have the same class label or the maximum depth required is reached.
- After the maximum depth, for the remaining data most common class label is assigned.
- For the test data traverse each tree from the root node and assign the class based on the **most common label** obtained after traversing each tree.

Bagging / Bootstrapping:

Bagging is used as ensemble algorithm to improve the accuracy over the base decision tree algorithm. This helps in avoiding the overfitting of the train data in a single model and employs model averaging by averaging the output.

Results:

	dataset1	dataset2
Accuracy	0.945488722	0.66457
Precision	0.948999671	0.640023
Recall	0.935638155	0.524041
F1measure	0.940686668	0.454022

Result Analysis:

- The performance of the Random forest is better than the Decision tree algorithm as decision tree overfits the training data. Thus ensemble of decision tree corrects the overfitting problem.

Cross Validation:

We have performed the cross validation in 10-fold on the Knn algorithm.

We first split the original data as the test data and train data for 10-fold validation and observer the resulting performance metrics.

For the next fold next set of test data and training data are obtained from the original data. The above procedure is repeated for 10 times to cover the entire data set.

Average accuracy is calculated from the above iterations.

Accuracy Results for cross validation:

Dataset1	Dataset2
0.912281	0.553191
0.982456	0.744681
0.964912	0.565217
0.947368	0.695652
0.947368	0.543478
0.947368	0.608696
0.947368	0.76087
0.982456	0.76087
0.894737	0.630435
0.928571	0.717391

Pros:

- Avoids overfitting problem of decision tree for the training data
- High accuracy compared to other base classification algorithms
- Suitable for high dimensional data as few features are considered randomly at a time
-

Cons:

- Computationally slower in testing phase compared to base decision tree algorithm as multiple trees need to be traversed to generate the class labels.
- One can expect inconsistent result over the iterations of the algorithm

Boosting

Algorithm:

- We have used **AdaBoost** technique for boosting.
- Initially assign equal weights to the training data w = 1/n, n = number of training data.
- We create a tree based on this trained data set using the decision tree algorithm
- For the test data traverse and classify them to compute the accuracy.
- Compute the **error** by summation of the weight of the misclassified points

$$\varepsilon_i = \frac{\sum_{j=1}^{N} w_j \delta(C_i(x_j) \neq y_j)}{\sum_{j=1}^{N} w_j}$$

- For every test data obtain the **prediction label** and assign alpha for that class from the above created decision trees and add them.
- Check the **maximum value of alpha** in the label to assign the class for the test data

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

- Based on the classification label assigned for the test data **update the weights** using the formula below.

$$w_j^{(i+1)} = \frac{w_j^{(i)} \exp\left(-\alpha_i y_j C_i(x_j)\right)}{Z^{(i)}}$$

We repeat the above procedure for all the trees. During each iteration we **increase the weights of the misclassified data** in the previous step and **decrease the weights of the correctly classified data.** This achieves more accuracy for the model.

Results:

	dataset1	dataset2
Accuracy	0.927882	0.642877
Precision	0.934973	0.582711
Recall	0.91418	0.528873
F1measure	0.919842	0.502782

Result Analysis:

- Overall performance is better than the decision tree.
- This is because the boosting assigns higher weights for the misclassified data and lower weights to the correctly classified.
- Due to the ensemble of trees the algorithm is relatively slow.

Cross Validation:

We have performed the cross validation in 10-fold on the Knn algorithm.

We first split the original data as the test data and train data for 10-fold validation and observer the resulting performance metrics.

For the next fold next set of test data and training data are obtained from the original data. The above procedure is repeated for 10 times to cover the entire data set.

Average accuracy is calculated from the above iterations.

Accuracy Results for cross validation:

Dataset1	Dataset2
0.929825	0.553191
0.912281	0.723404
0.982456	0.673913
0.982456	0.652174
0.912281	0.565217
0.912281	0.565217
0.929825	0.695652
0.947368	0.782609
0.877193	0.630435
0.892857	0.586957

Pros:

- Greater Accuracy over decision tree algorithm
- Faster compared to other classification algorithm
- Final model converges to be a stronger learner than the random guessing
- Less susceptible to overfitting compared to other algorithms

Cons:

- Sensitive to noise and outliers
- May require more iterations for converging