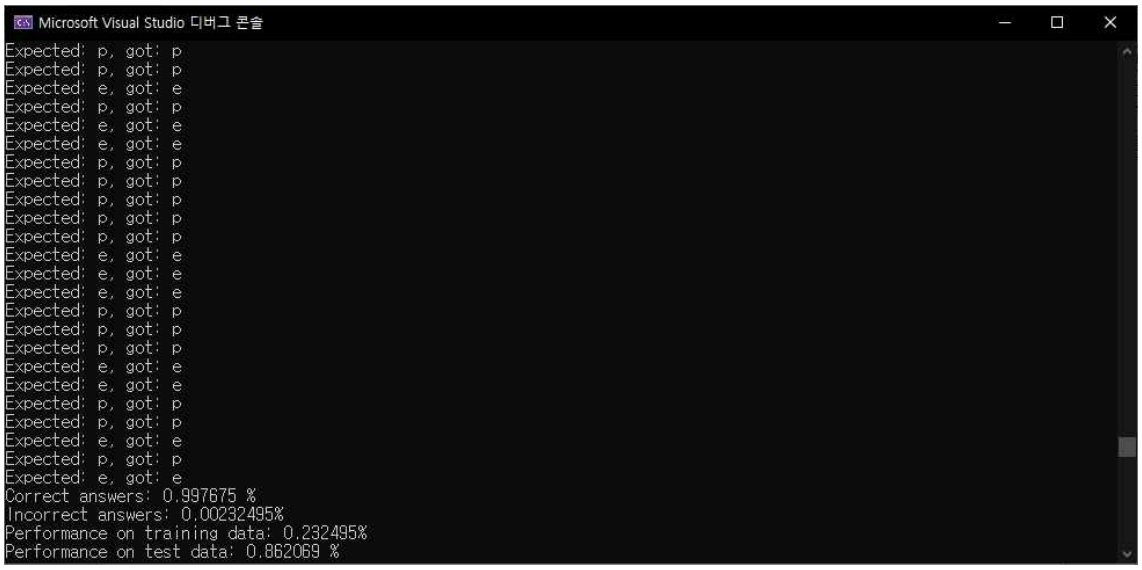
Decision Tree.pdf의 15~17 페이지를 참고하여 구현 및 분류 결과 확인

```
int mushrooms() {
	// If the caller gave a filename, great. Otherwise, use a default.
	//
	//const char *csv_file_name = argc >= 2 ? argv[1] : 
"../mushroom/agaricus-lepiota.data";
	const char* csv_file_name = "agaricus-lepiota.data";
	cout << "OpenCV Version: " << CV_VERSION << endl;


	// Read in the CSV file that we were given.
	//
	cv::Ptr<cv::ml::TrainData> data_set =
		cv::ml::TrainData::loadFromCSV(csv_file_name, // Input file name
			0, // Header lines (ignore this many)
			0, // Responses are (start) at thie column
			1, // Inputs start at this column
			"cat[0-22]" // All 23 columns are categorical
		);
	// Use defaults for delimeter (',') and missch ('?')
	// Verify that we read in what we think.
	//
	int n_samples = data_set->getNSamples();
	if (n_samples == 0) {
		cerr << "Could not read file: " << csv_file_name << endl;
		//exit(-1);
	}
	else {
		cout << "Read " << n_samples << " samples from " << csv_file_name
<< endl;
	}

	// Split the data, so that 90% is train data
	//
	data_set->setTrainTestSplitRatio(0.90, false);
	int n_train_samples = data_set->getNTrainSamples();
	int n_test_samples = data_set->getNTestSamples();
	cout << "Found " << n_train_samples << " Train Samples, and "
		<< n_test_samples << " Test Samples" << endl;
```

```
// Create a DTrees classifier.
//
cv::Ptr<cv::ml::RTrees> dtree = cv::ml::RTrees::create();
// set parameters
//
// These are the parameters from the old mushrooms.cpp code
// Set up priors to penalize "poisonous" 10x as much as "edible"
//
float _priors[] = { 1.0, 10.0 };
cv::Mat priors(1, 2, CV_32F, _priors);
dtree->setMaxDepth(8);
dtree->setMinSampleCount(10);
dtree->setRegressionAccuracy(0.01f);
dtree->setUseSurrogates(false /* true */);
dtree->setMaxCategories(15);
dtree->setCVFolds(0 /*10*/); // nonzero causes core dump
dtree->setUse1SERule(true);
dtree->setTruncatePrunedTree(true);
dtree->setPriors(priors);
//dtree->setPriors(cv::Mat()); // ignore priors for now...
// Now train the model
// NB: we are only using the "train" part of the data set
//
dtree->train(data_set);

// Having successfully trained the data, we should be able
// to calculate the error on both the training data, as well
// as the test data that we held out.
//
cv::Mat results;
float train_performance = dtree->calcError(data_set,
        false, // use train data
        results // cv::noArray()
);
std::vector<cv::String> names;
data_set->getNames(names);
Mat flags = data_set->getVarSymbolFlags();

// Compute some statistics on our own:
//
{
```

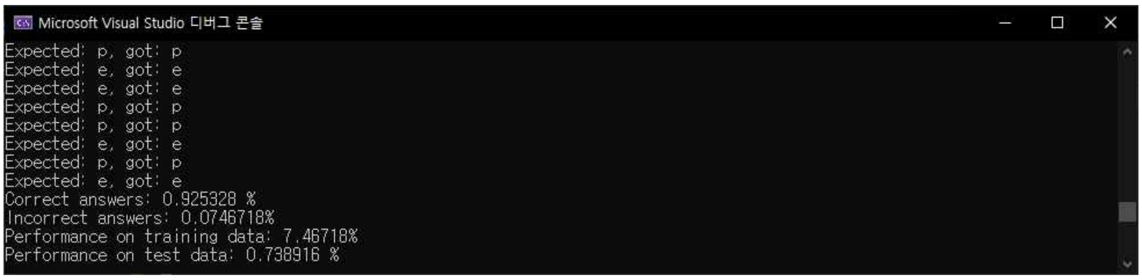
```
                cv::Mat expected_responses = data_set->getResponses();
                int good = 0, bad = 0, total = 0;
                for (int i = 0; i < data_set->getNTrainSamples(); ++i) {
                        float received = results.at<float>(i, 0);
                        float expected = expected_responses.at<float>(i, 0);
                        cv::String r_str = names[(int)received];
                        cv::String e_str = names[(int)expected];
                        cout << "Expected: " << e_str << ", got: " << r_str << endl;
                        if (received == expected)
                                good++;
                        else
                                bad++;
                        total++;
                }
                cout << "Correct answers: " << (float(good) / total) << " % " << endl;
                cout << "Incorrect answers: " << (float(bad) / total) << "%"
                        << endl;
        }
        float test_performance = dtree->calcError(data_set,
                true, // use test data
                results // cv::noArray()
        );
        cout << "Performance on training data: " << train_performance << "%" <<
endl;
        cout << "Performance on test data: " << test_performance << " % " << endl;

        waitKey(0);
        return 0;
}
```

실행결과


Microsoft Visual Studio 디버그 콘솔

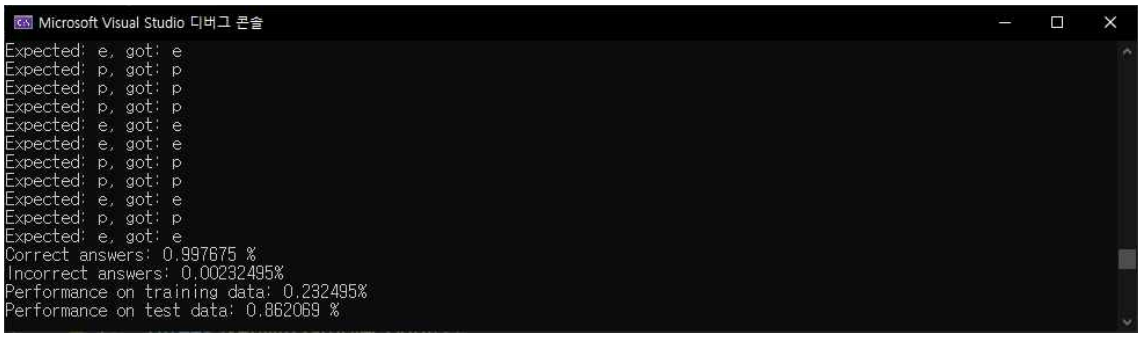
```
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: p, got: p
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: e, got: e
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.997675 %
Incorrect answers: 0.00232495%
Performance on training data: 0.232495%
Performance on test data: 0.862069 %
```

setMaxDepth를 8=>3으로 변경



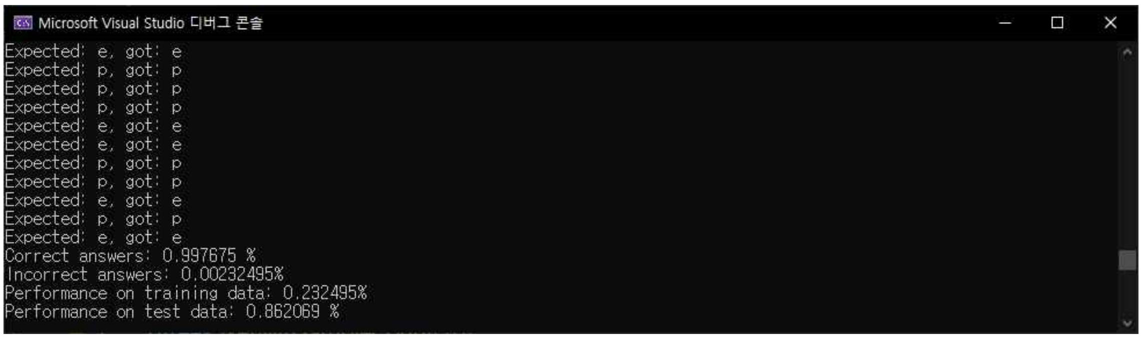
```
Expected: p, got: p
Expected: e, got: e
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.925328 %
Incorrect answers: 0.0746718%
Performance on training data: 7.46718%
Performance on test data: 0.738916 %
```

setMinSampleCount를 10=>2로 변경



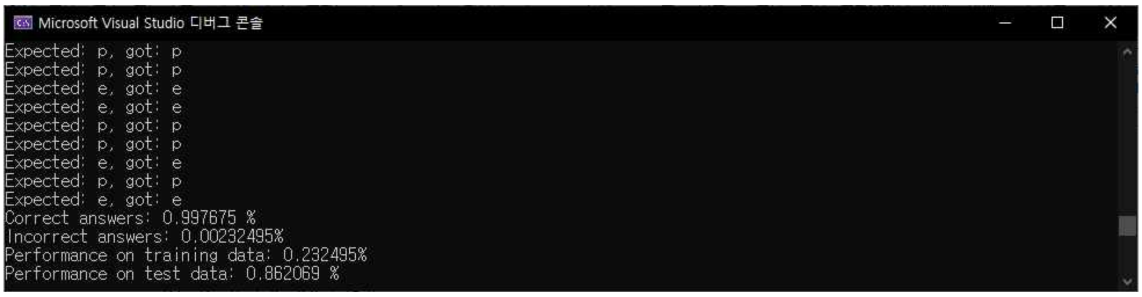
```
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.997675 %
Incorrect answers: 0.00232495%
Performance on training data: 0.232495%
Performance on test data: 0.862069 %
```

setRegressionAccuracy를 0.01f=>0.5f로 변경



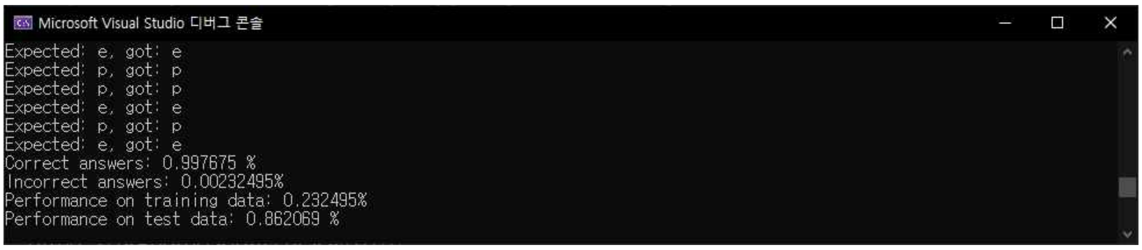
```
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.997675 %
Incorrect answers: 0.00232495%
Performance on training data: 0.232495%
Performance on test data: 0.862069 %
```

setMaxCategories를 15=>3으로 변경



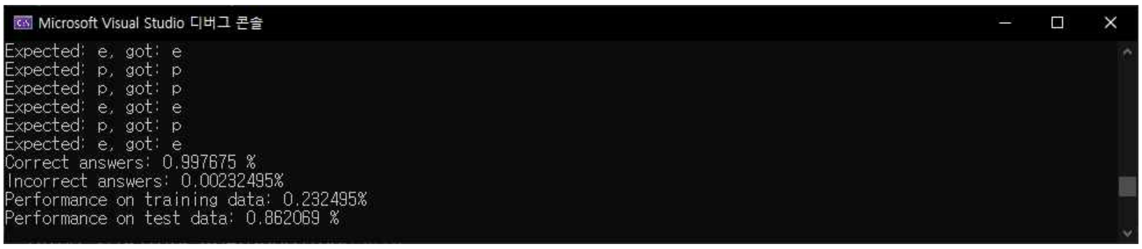
```
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.997675 %
Incorrect answers: 0.00232495%
Performance on training data: 0.232495%
Performance on test data: 0.862069 %
```

setUse1SERule를 true=>false로 변경



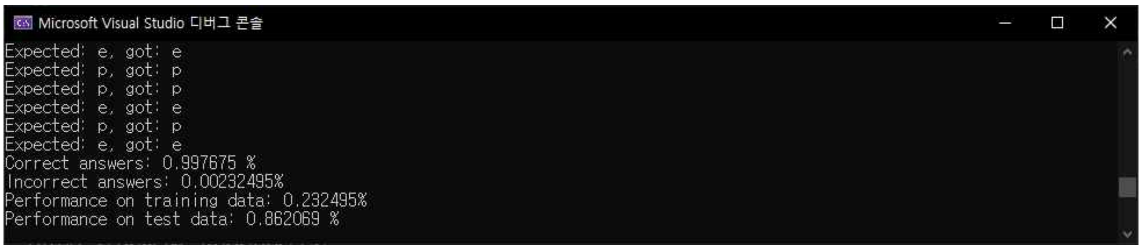
```
Microsoft Visual Studio 디버그 콘솔
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.997675 %
Incorrect answers: 0.00232495%
Performance on training data: 0.232495%
Performance on test data: 0.862069 %
```

setTruncatePrunedTree를 true=>false로 변경



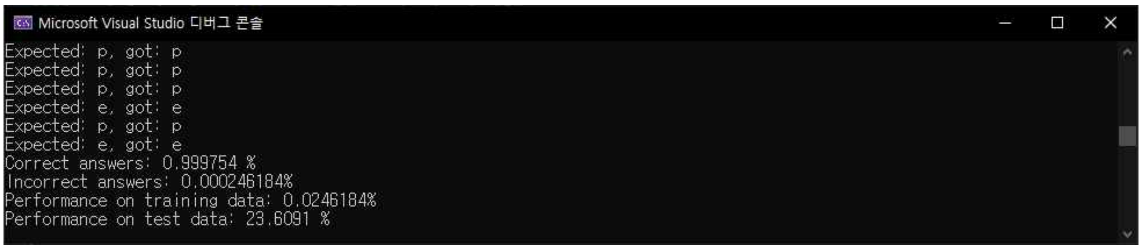
```
Microsoft Visual Studio 디버그 콘솔
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.997675 %
Incorrect answers: 0.00232495%
Performance on training data: 0.232495%
Performance on test data: 0.862069 %
```

float _priors[] = { 1.0, 10.0 }; => float _priors[] = { 1.0, 5.0 };로 변경



```
Microsoft Visual Studio 디버그 콘솔
Expected: e, got: e
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.997675 %
Incorrect answers: 0.00232495%
Performance on training data: 0.232495%
Performance on test data: 0.862069 %
```

data_set->setTrainTestSplitRatio(0.90, false);
=> data_set->setTrainTestSplitRatio(0.50, false);로 변경



```
Microsoft Visual Studio 디버그 콘솔
Expected: p, got: p
Expected: p, got: p
Expected: p, got: p
Expected: e, got: e
Expected: p, got: p
Expected: e, got: e
Correct answers: 0.999754 %
Incorrect answers: 0.000246184%
Performance on training data: 0.0246184%
Performance on test data: 23.6091 %
```

max depth를 변경했을 때 정확도가 다소 낮아짐을 제외하고는 결과에 큰 영향을 끼치지 않음을 알 수 있다.