2주 5일차

분류 및 머신러닝 기본

강의 내용

Contents of Lecture

기간	내용	실습
1주 1일차	Ipython 기초	문제 및 답안 코드 제공
1주 2일차	Numpy	문제 및 답안 코드 제공
1주 3일차	Pandas	문제 및 답안 코드 제공
1주 4일차	Matplotlib	문제 및 답안 코드 제공
1주 5일차	EDA	문제 및 답안 코드 제공
2주 1일차	통계 분포	문제 및 답안 코드 제공
2주 2일차	통계적 실험 & 유의성 검정	문제 및 답안 코드 제공
2주 3일차	인공지능 개념 & 예측모형	문제 및 답안 코드 제공
2주 4일차	회귀 및 분류	문제 및 답안 코드 제공
2주 5일차	**분류 & 머신러닝 기본**	**문제 및 답안 코드 제공**

기간	내용	과제
3주 1일차	머신러닝 응용 및 고급	문제 및 답안 코드 제공
3주 2일차	딥러닝 기본 & 응용	문제 및 답안 코드 제공
3주 3일차	고급 딥러닝	문제 및 답안 코드 제공
3주 4일차	알고보면 쓸모있는 신기한 기계학습	문제 및 답안 코드 제공
3주 5일차	MLOps란?	문제 및 답안 코드 제공
4주 1일차	MLOps 준비	문제 및 답안 코드 제공
4주 2일차	MLOps 구축 3단계 및 파이프라인 구축	문제 및 답안 코드 제공
4주 3일차	MLFlow 소개 및 활용	문제 및 답안 코드 제공
4주 4일차	AWS/Azure 배포	문제 및 답안 코드 제공
4주 5일차	Google 배포 및 Databricks 활용 & 최종 개별 프로젝트 발표	실습코드 제공 프로젝트 발표

1부: 분류

1) 분류 모델 평가하기

2) 불균형 데이터 다루기

분류 모델 평가하기

Evaluating Classification Models

Evaluating Classification Models

It is common in predictive modeling to train a number of different models, apply each to a holdout sample, and assess their performance. Sometimes, after a number of models have been evaluated and tuned, and if there are enough data, a third holdout sample, not used previously, is used to estimate how the chosen model will perform with completely new data. Different disciplines and practitioners will also use the terms *validation* and *test* to refer to the holdout sample(s). Fundamentally, the assessment process attempts to learn which model produces the most accurate and useful predictions.

A simple way to measure classification performance is to count the proportion of predictions that are correct, i.e., measure the *accuracy*. Accuracy is simply a measure of total error:

$$\text{accuracy} = \frac{\sum \text{TruePositive} + \sum \text{TrueNegative}}{\text{SampleSize}}$$

In most classification algorithms, each case is assigned an "estimated probability of being a 1."[3] The default decision point, or cutoff, is typically 0.50 or 50%. If the probability is above 0.5, the classification is "1"; otherwise it is "0." An alternative default cutoff is the prevalent probability of 1s in the data.

Key Terms for Evaluating Classification Models

Accuracy
: The percent (or proportion) of cases classified correctly.

Confusion matrix
: A tabular display (2×2 in the binary case) of the record counts by their predicted and actual classification status.

Sensitivity
: The percent (or proportion) of all 1s that are correctly classified as 1s.

Synonym
: Recall

Specificity
: The percent (or proportion) of all 0s that are correctly classified as 0s.

Precision
: The percent (proportion) of predicted 1s that are actually 1s.

ROC curve
: A plot of sensitivity versus specificity.

Lift
: A measure of how effective the model is at identifying (comparatively rare) 1s at different probability cutoffs.

분류 모델 평가하기 > Confusion Matrix

Evaluating Classification Models

Confusion Matrix

At the heart of classification metrics is the *confusion matrix*. The confusion matrix is a table showing the number of correct and incorrect predictions categorized by type of response. Several packages are available in *R* and *Python* to compute a confusion matrix, but in the binary case, it is simple to compute one by hand.

To illustrate the confusion matrix, consider the `logistic_gam` model that was trained on a balanced data set with an equal number of defaulted and paid-off loans (see Figure 5-4). Following the usual conventions, $Y = 1$ corresponds to the event of interest (e.g., default), and $Y = 0$ corresponds to a negative (or usual) event (e.g., paid off). The following computes the confusion matrix for the `logistic_gam` model applied to the entire (unbalanced) training set in *R*:

```
pred <- predict(logistic_gam, newdata=train_set)
pred_y <- as.numeric(pred > 0)
true_y <- as.numeric(train_set$outcome=='default')
true_pos <- (true_y==1) & (pred_y==1)
true_neg <- (true_y==0) & (pred_y==0)
false_pos <- (true_y==0) & (pred_y==1)
false_neg <- (true_y==1) & (pred_y==0)
conf_mat <- matrix(c(sum(true_pos), sum(false_pos),
                     sum(false_neg), sum(true_neg)), 2, 2)
colnames(conf_mat) <- c('Yhat = 1', 'Yhat = 0')
rownames(conf_mat) <- c('Y = 1', 'Y = 0')
conf_mat
      Yhat = 1 Yhat = 0
Y = 1 14295    8376
Y = 0 8052     14619
```

In *Python*:

```
pred = logit_reg.predict(X)
pred_y = logit_reg.predict(X) == 'default'
true_y = y == 'default'
true_pos = true_y & pred_y
true_neg = ~true_y & ~pred_y
false_pos = ~true_y & pred_y
false_neg = true_y & ~pred_y
conf_mat = pd.DataFrame([[np.sum(true_pos), np.sum(false_neg)],
                         [np.sum(false_pos), np.sum(true_neg)]],
                        index=['Y = default', 'Y = paid off'],
                        columns=['Yhat = default', 'Yhat = paid off'])
conf_mat
```

The predicted outcomes are columns and the true outcomes are the rows. The diagonal elements of the matrix show the number of correct predictions, and the off-diagonal elements show the number of incorrect predictions. For example, 14,295 defaulted loans were correctly predicted as a default, but 8,376 defaulted loans were incorrectly predicted as paid off.

분류 모델 평가하기 > Confusion Matrix

Evaluating Classification Models

Figure 5-5 shows the relationship between the confusion matrix for a binary response Y and different metrics (see "Precision, Recall, and Specificity" on page 223 for more on the metrics). As with the example for the loan data, the actual response is along the rows and the predicted response is along the columns. The diagonal boxes (upper left, lower right) show when the predictions $\hat{Y}$ correctly predict the response. One important metric not explicitly called out is the false positive *rate* (the mirror image of precision). When 1s are rare, the ratio of false positives to all predicted positives can be high, leading to the unintuitive situation in which a predicted 1 is most likely a 0. This problem plagues medical screening tests (e.g., mammograms) that are widely applied: due to the relative rarity of the condition, positive test results most likely do not mean breast cancer. This leads to much confusion in the public.

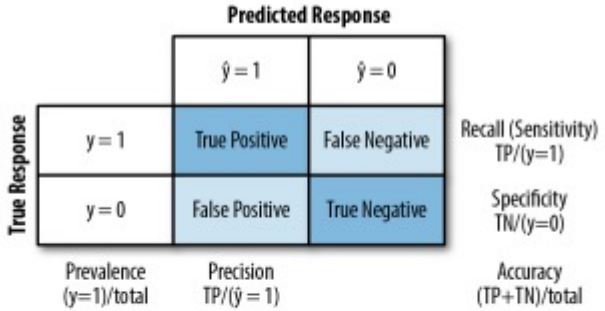


Figure 5-5. Confusion matrix for a binary response and various metrics

Here, we present the actual response along the rows and the predicted response along the columns, but it is not uncommon to see this reversed. A notable example is the popular `caret` package in *R*.

분류 모델 평가하기 > The Rare Class Problem

The Rare Class Problem

In many cases, there is an imbalance in the classes to be predicted, with one class much more prevalent than the other—for example, legitimate insurance claims versus fraudulent ones, or browsers versus purchasers at a website. The rare class (e.g., the fraudulent claims) is usually the class of more interest and is typically designated 1, in contrast to the more prevalent 0s. In the typical scenario, the 1s are the more important case, in the sense that misclassifying them as 0s is costlier than misclassifying 0s as 1s. For example, correctly identifying a fraudulent insurance claim may save thousands of dollars. On the other hand, correctly identifying a nonfraudulent claim merely saves you the cost and effort of going through the claim by hand with a more careful review (which is what you would do if the claim were tagged as "fraudulent").

In such cases, unless the classes are easily separable, the most accurate classification model may be one that simply classifies everything as a 0. For example, if only 0.1% of the browsers at a web store end up purchasing, a model that predicts that each browser will leave without purchasing will be 99.9% accurate. However, it will be useless. Instead, we would be happy with a model that is less accurate overall but is good at picking out the purchasers, even if it misclassifies some nonpurchasers along the way.

Precision, Recall, and Specificity

Metrics other than pure accuracy—metrics that are more nuanced—are commonly used in evaluating classification models. Several of these have a long history in statistics—especially biostatistics, where they are used to describe the expected performance of diagnostic tests. The *precision* measures the accuracy of a predicted positive outcome (see Figure 5-5):

$$\text{precision} = \frac{\sum \text{TruePositive}}{\sum \text{TruePositive} + \sum \text{FalsePositive}}$$

The *recall*, also known as *sensitivity*, measures the strength of the model to predict a positive outcome—the proportion of the 1s that it correctly identifies (see Figure 5-5). The term *sensitivity* is used a lot in biostatistics and medical diagnostics, whereas *recall* is used more in the machine learning community. The definition of recall is:

$$\text{recall} = \frac{\sum \text{TruePositive}}{\sum \text{TruePositive} + \sum \text{FalseNegative}}$$

Another metric used is *specificity*, which measures a model's ability to predict a negative outcome:

$$\text{specificity} = \frac{\sum \text{TrueNegative}}{\sum \text{TrueNegative} + \sum \text{FalsePositive}}$$

We can calculate the three metrics from `conf_mat` in *R*:

```
# precision
conf_mat[1, 1] / sum(conf_mat[,1])
# recall
conf_mat[1, 1] / sum(conf_mat[1,])
# specificity
conf_mat[2, 2] / sum(conf_mat[2,])
```

Here is the equivalent code to calculate the metrics in *Python*:

```
conf_mat = confusion_matrix(y, logit_reg.predict(X))
print('Precision', conf_mat[0, 0] / sum(conf_mat[:, 0]))
print('Recall', conf_mat[0, 0] / sum(conf_mat[0, :]))
print('Specificity', conf_mat[1, 1] / sum(conf_mat[1, :]))

precision_recall_fscore_support(y, logit_reg.predict(X),
                                labels=['default', 'paid off'])
```

`scikit-learn` has a custom method `precision_recall_fscore_support` that calculates precision and recall/specificity all at once.

ROC Curve

You can see that there is a trade-off between recall and specificity. Capturing more 1s generally means misclassifying more 0s as 1s. The ideal classifier would do an excellent job of classifying the 1s, without misclassifying more 0s as 1s.

The metric that captures this trade-off is the "Receiver Operating Characteristics" curve, usually referred to as the *ROC curve*. The ROC curve plots recall (sensitivity) on the y-axis against specificity on the x-axis.[4] The ROC curve shows the trade-off between recall and specificity as you change the cutoff to determine how to classify a record. Sensitivity (recall) is plotted on the y-axis, and you may encounter two forms in which the x-axis is labeled:

- Specificity plotted on the x-axis, with 1 on the left and 0 on the right
- 1-Specificity plotted on the x-axis, with 0 on the left and 1 on the right

The curve looks identical whichever way it is done. The process to compute the ROC curve is:

1. Sort the records by the predicted probability of being a 1, starting with the most probable and ending with the least probable.
2. Compute the cumulative specificity and recall based on the sorted records.

Computing the ROC curve in *R* is straightforward. The following code computes ROC for the loan data:

```
idx <- order(-pred)
recall <- cumsum(true_y[idx] == 1) / sum(true_y == 1)
specificity <- (sum(true_y == 0) - cumsum(true_y[idx] == 0)) / sum(true_y == 0)
roc_df <- data.frame(recall = recall, specificity = specificity)
ggplot(roc_df, aes(x=specificity, y=recall)) +
  geom_line(color='blue') +
  scale_x_reverse(expand=c(0, 0)) +
  scale_y_continuous(expand=c(0, 0)) +
  geom_line(data=data.frame(x=(0:100) / 100), aes(x=x, y=1-x),
            linetype='dotted', color='red')
```

In *Python*, we can use the `scikit-learn` function `sklearn.metrics.roc_curve` to calculate the required information for the ROC curve. You can find similar packages for *R*, e.g., `ROCR`:

```
fpr, tpr, thresholds = roc_curve(y, logit_reg.predict_proba(X)[:,0],
                                 pos_label='default')
roc_df = pd.DataFrame({'recall': tpr, 'specificity': 1 - fpr})

ax = roc_df.plot(x='specificity', y='recall', figsize=(4, 4), legend=False)
ax.set_ylim(0, 1)
ax.set_xlim(1, 0)
ax.plot((1, 0), (0, 1))
ax.set_xlabel('specificity')
ax.set_ylabel('recall')
```

The result is shown in Figure 5-6. The dotted diagonal line corresponds to a classifier no better than random chance. An extremely effective classifier (or, in medical situations, an extremely effective diagnostic test) will have an ROC that hugs the upper-left corner—it will correctly identify lots of 1s without misclassifying lots of 0s as 1s. For this model, if we want a classifier with a specificity of at least 50%, then the recall is about 75%.

분류 모델 평가하기 > ROC Curve

Evaluating Classification Models

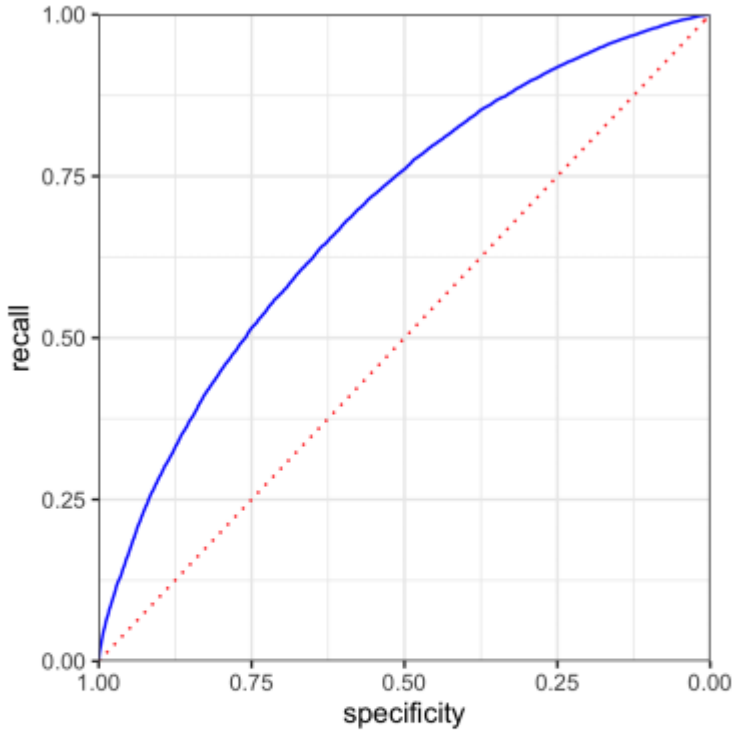


Figure 5-6. ROC curve for the loan data

Precision-Recall Curve

In addition to ROC curves, it can be illuminating to examine the precision-recall (PR) curve. PR curves are computed in a similar way except that the data is ordered from least to most probable and cumulative precision and recall statistics are computed. PR curves are especially useful in evaluating data with highly unbalanced outcomes.

분류 모델 평가하기 > AUC

Evaluating Classification Models

AUC

The ROC curve is a valuable graphical tool, but by itself doesn't constitute a single measure for the performance of a classifier. The ROC curve can be used, however, to produce the area underneath the curve (AUC) metric. AUC is simply the total area under the ROC curve. The larger the value of AUC, the more effective the classifier. An AUC of 1 indicates a perfect classifier: it gets all the 1s correctly classified, and it doesn't misclassify any 0s as 1s.

A completely ineffective classifier—the diagonal line—will have an AUC of 0.5.

Figure 5-7 shows the area under the ROC curve for the loan model. The value of AUC can be computed by a numerical integration in *R*:

```
sum(roc_df$recall[-1] * diff(1 - roc_df$specificity))
    [1] 0.6926172
```

In *Python*, we can either calculate the accuracy as shown for *R* or use `scikit-learn`'s function `sklearn.metrics.roc_auc_score`. You will need to provide the expected value as 0 or 1:

```
print(np.sum(roc_df.recall[:-1] * np.diff(1 - roc_df.specificity)))
print(roc_auc_score([1 if yi == 'default' else 0 for yi in y],
                    logit_reg.predict_proba(X)[:, 0]))
```

The model has an AUC of about 0.69, corresponding to a relatively weak classifier.

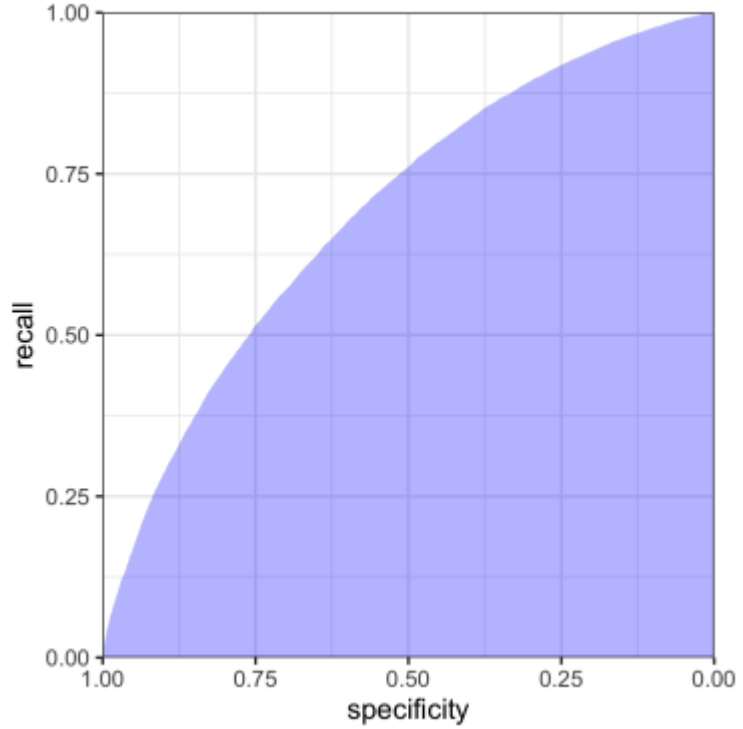


Figure 5-7. Area under the ROC curve for the loan data

False Positive Rate Confusion

False positive/negative rates are often confused or conflated with specificity or sensitivity (even in publications and software!). Sometimes the false positive rate is defined as the proportion of true negatives that test positive. In many cases (such as network intrusion detection), the term is used to refer to the proportion of positive signals that are true negatives.

Lift

Using the AUC as a metric to evaluate a model is an improvement over simple accuracy, as it can assess how well a classifier handles the trade-off between overall accuracy and the need to identify the more important 1s. But it does not completely address the rare-case problem, where you need to lower the model's probability cutoff below 0.5 to avoid having all records classified as 0. In such cases, for a record to be classified as a 1, it might be sufficient to have a probability of 0.4, 0.3, or lower. In effect, we end up overidentifying 1s, reflecting their greater importance.

Changing this cutoff will improve your chances of catching the 1s (at the cost of misclassifying more 0s as 1s). But what is the optimum cutoff?

The concept of lift lets you defer answering that question. Instead, you consider the records in order of their predicted probability of being 1s. Say, of the top 10% classified as 1s, how much better did the algorithm do, compared to the benchmark of simply picking blindly? If you can get 0.3% response in this top decile instead of the 0.1% you get overall by picking randomly, the algorithm is said to have a *lift* (also called *gains*) of 3 in the top decile. A lift chart (gains chart) quantifies this over the range of the data. It can be produced decile by decile, or continuously over the range of the data.

To compute a lift chart, you first produce a *cumulative gains chart* that shows the recall on the y-axis and the total number of records on the x-axis. The *lift curve* is the ratio of the cumulative gains to the diagonal line corresponding to random selection. *Decile gains charts* are one of the oldest techniques in predictive modeling, dating from the days before internet commerce. They were particularly popular among direct mail professionals. Direct mail is an expensive method of advertising if applied indiscriminately, and advertisers used predictive models (quite simple ones, in the early days) to identify the potential customers with the likeliest prospect of payoff.

Uplift

Sometimes the term *uplift* is used to mean the same thing as lift. An alternate meaning is used in a more restrictive setting, when an A/B test has been conducted and the treatment (A or B) is then used as a predictor variable in a predictive model. The uplift is the improvement in response predicted *for an individual case* with treatment A versus treatment B. This is determined by scoring the individual case first with the predictor set to A, and then again with the predictor toggled to B. Marketers and political campaign consultants use this method to determine which of two messaging treatments should be used with which customers or voters.

분류 모델 평가하기 > Lift

Evaluating Classification Models

A lift curve lets you look at the consequences of setting different probability cutoffs for classifying records as 1s. It can be an intermediate step in settling on an appropriate cutoff level. For example, a tax authority might have only a certain amount of resources that it can spend on tax audits, and it wants to spend them on the likeliest tax cheats. With its resource constraint in mind, the authority would use a lift chart to estimate where to draw the line between tax returns selected for audit and those left alone.

Key Ideas

- Accuracy (the percent of predicted classifications that are correct) is but a first step in evaluating a model.
- Other metrics (recall, specificity, precision) focus on more specific performance characteristics (e.g., recall measures how good a model is at correctly identifying 1s).
- AUC (area under the ROC curve) is a common metric for the ability of a model to distinguish 1s from 0s.
- Similarly, lift measures how effective a model is in identifying the 1s, and it is often calculated decile by decile, starting with the most probable 1s.

Strategies for Imbalanced Data

The previous section dealt with evaluation of classification models using metrics that go beyond simple accuracy and are suitable for imbalanced data—data in which the outcome of interest (purchase on a website, insurance fraud, etc.) is rare. In this section, we look at additional strategies that can improve predictive modeling performance with imbalanced data.

Key Terms for Imbalanced Data

Undersample
Use fewer of the prevalent class records in the classification model.

Synonym
Downsample

Oversample
Use more of the rare class records in the classification model, bootstrapping if necessary.

Synonym
Upsample

Up weight or down weight
Attach more (or less) weight to the rare (or prevalent) class in the model.

Data generation
Like bootstrapping, except each new bootstrapped record is slightly different from its source.

z-score
The value that results after standardization.

K
The number of neighbors considered in the nearest neighbor calculation.

불균형 데이터 다루기 > Undersampling

Undersampling

If you have enough data, as is the case with the loan data, one solution is to *undersample* (or downsample) the prevalent class, so the data to be modeled is more balanced between 0s and 1s. The basic idea in undersampling is that the data for the dominant class has many redundant records. Dealing with a smaller, more balanced data set yields benefits in model performance, and it makes it easier to prepare the data and to explore and pilot models.

How much data is enough? It depends on the application, but in general, having tens of thousands of records for the less dominant class is enough. The more easily distinguishable the 1s are from the 0s, the less data needed.

The loan data analyzed in "Logistic Regression" on page 208 was based on a balanced training set: half of the loans were paid off, and the other half were in default. The predicted values were similar: half of the probabilities were less than 0.5, and half were greater than 0.5. In the full data set, only about 19% of the loans were in default, as shown in *R*:

```
mean(full_train_set$outcome=='default')
[1] 0.1889455
```

In *Python*:

```
print('percentage of loans in default: ',
      100 * np.mean(full_train_set.outcome == 'default'))
```

What happens if we use the full data set to train the model? Let's see what this looks like in *R*:

```
full_model <- glm(outcome ~ payment_inc_ratio + purpose_ + home_ +
                            emp_len_ + dti + revol_bal + revol_util,
                  data=full_train_set, family='binomial')
pred <- predict(full_model)
mean(pred > 0)
[1] 0.003942094
```

And in *Python*:

```
predictors = ['payment_inc_ratio', 'purpose_', 'home_', 'emp_len_',
              'dti', 'revol_bal', 'revol_util']
outcome = 'outcome'
X = pd.get_dummies(full_train_set[predictors], prefix='', prefix_sep='',
                   drop_first=True)
y = full_train_set[outcome]

full_model = LogisticRegression(penalty='l2', C=1e42, solver='liblinear')
full_model.fit(X, y)
print('percentage of loans predicted to default: ',
      100 * np.mean(full_model.predict(X) == 'default'))
```

Only 0.39% of the loans are predicted to be in default, or less than 1/47 of the expected number.[5] The loans that were paid off overwhelm the loans in default because the model is trained using all the data equally. Thinking about it intuitively, the presence of so many nondefaulting loans, coupled with the inevitable variability in predictor data, means that, even for a defaulting loan, the model is likely to find some nondefaulting loans that it is similar to, by chance. When a balanced sample was used, roughly 50% of the loans were predicted to be in default.

Oversampling and Up/Down Weighting

One criticism of the undersampling method is that it throws away data and is not using all the information at hand. If you have a relatively small data set, and the rarer class contains a few hundred or a few thousand records, then undersampling the dominant class has the risk of throwing out useful information. In this case, instead of downsampling the dominant case, you should oversample (upsample) the rarer class by drawing additional rows with replacement (bootstrapping).

You can achieve a similar effect by weighting the data. Many classification algorithms take a weight argument that will allow you to up/down weight the data. For example, apply a weight vector to the loan data using the `weight` argument to `glm` in *R*:

```
wt <- ifelse(full_train_set$outcome=='default',
             1 / mean(full_train_set$outcome == 'default'), 1)
full_model <- glm(outcome ~ payment_inc_ratio + purpose_ + home_ +
                            emp_len_ + dti + revol_bal + revol_util,
                  data=full_train_set, weight=wt, family='quasibinomial')
pred <- predict(full_model)
mean(pred > 0)
[1] 0.5767208
```

Most `scikit-learn` methods allow specifying weights in the `fit` function using the keyword argument `sample_weight`:

```
default_wt = 1 / np.mean(full_train_set.outcome == 'default')
wt = [default_wt if outcome == 'default' else 1
      for outcome in full_train_set.outcome]

full_model = LogisticRegression(penalty="l2", C=1e42, solver='liblinear')
full_model.fit(X, y, sample_weight=wt)
print('percentage of loans predicted to default (weighting): ',
      100 * np.mean(full_model.predict(X) == 'default'))
```

The weights for loans that default are set to $\frac{1}{p}$, where p is the probability of default. The nondefaulting loans have a weight of 1. The sums of the weights for the defaulting loans and nondefaulting loans are roughly equal. The mean of the predicted values is now about 58% instead of 0.39%.

Note that weighting provides an alternative to both upsampling the rarer class and downsampling the dominant class.

Adapting the Loss Function

Many classification and regression algorithms optimize a certain criteria or *loss function*. For example, logistic regression attempts to minimize the deviance. In the literature, some propose to modify the loss function in order to avoid the problems caused by a rare class. In practice, this is hard to do: classification algorithms can be complex and difficult to modify. Weighting is an easy way to change the loss function, discounting errors for records with low weights in favor of records with higher weights.

불균형 데이터 다루기 > Data Generation

Data Generation

A variation of upsampling via bootstrapping (see "Oversampling and Up/Down Weighting" on page 232) is *data generation* by perturbing existing records to create new records. The intuition behind this idea is that since we observe only a limited set of instances, the algorithm doesn't have a rich set of information to build classification "rules." By creating new records that are similar but not identical to existing records, the algorithm has a chance to learn a more robust set of rules. This notion is similar in spirit to ensemble statistical models such as boosting and bagging (see Chapter 6).

The idea gained traction with the publication of the *SMOTE* algorithm, which stands for "Synthetic Minority Oversampling Technique." The SMOTE algorithm finds a record that is similar to the record being upsampled (see "K-Nearest Neighbors" on page 238) and creates a synthetic record that is a randomly weighted average of the original record and the neighboring record, where the weight is generated separately for each predictor. The number of synthetic oversampled records created depends on the oversampling ratio required to bring the data set into approximate balance with respect to outcome classes.

There are several implementations of SMOTE in *R*. The most comprehensive package for handling unbalanced data is `unbalanced`. It offers a variety of techniques, including a "Racing" algorithm to select the best method. However, the SMOTE algorithm is simple enough that it can be implemented directly in *R* using the `FNN` package.

The *Python* package `imbalanced-learn` implements a variety of methods with an API that is compatible with `scikit-learn`. It provides various methods for over- and undersampling and support for using these techniques with boosting and bagging classifiers.

불균형 데이터 다루기 > Cost-Based Classification

Cost-Based Classification

In practice, accuracy and AUC are a poor man's way to choose a classification rule. Often, an estimated cost can be assigned to false positives versus false negatives, and it is more appropriate to incorporate these costs to determine the best cutoff when classifying 1s and 0s. For example, suppose the expected cost of a default of a new loan is C and the expected return from a paid-off loan is R. Then the expected return for that loan is:

$$\text{expected return} = P(Y = 0) \times R + P(Y = 1) \times C$$

Instead of simply labeling a loan as default or paid off, or determining the probability of default, it makes more sense to determine if the loan has a positive expected return. Predicted probability of default is an intermediate step, and it must be combined with the loan's total value to determine expected profit, which is the ultimate planning metric of business. For example, a smaller value loan might be passed over in favor of a larger one with a slightly higher predicted default probability.

Exploring the Predictions

A single metric, such as AUC, cannot evaluate all aspects of the suitability of a model for a situation. Figure 5-8 displays the decision rules for four different models fit to the loan data using just two predictor variables: `borrower_score` and `payment_inc_ratio`. The models are linear discriminant analysis (LDA), logistic linear regression, logistic regression fit using a generalized additive model (GAM), and a tree model (see "Tree Models" on page 249). The region to the upper left of the lines corresponds to a predicted default. LDA and logistic linear regression give nearly identical results in this case. The tree model produces the least regular rule, with two steps. Finally, the GAM fit of the logistic regression represents a compromise between the tree model and the linear model.

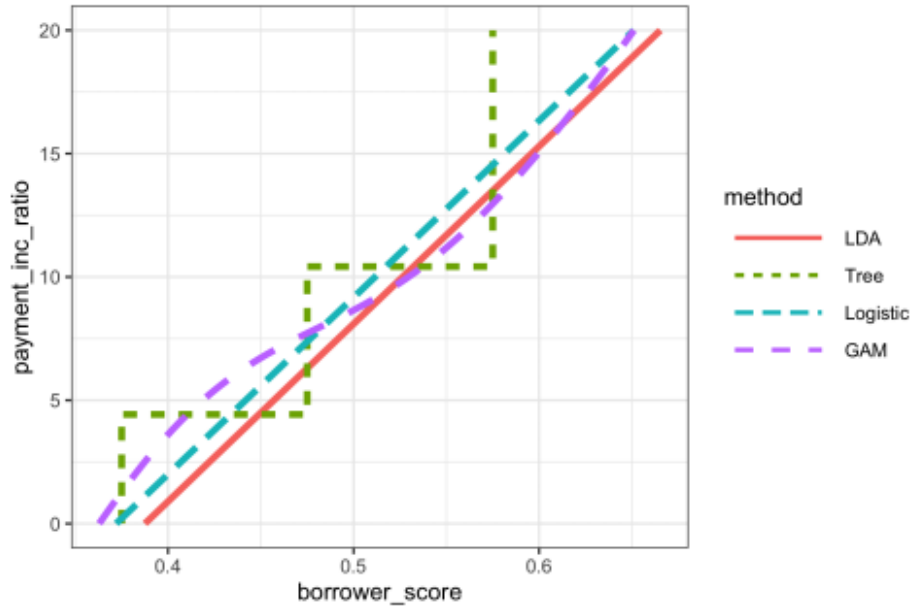


Figure 5-8. Comparison of the classification rules for four different methods

It is not easy to visualize the prediction rules in higher dimensions or, in the case of the GAM and the tree model, even to generate the regions for such rules.

In any case, exploratory analysis of predicted values is always warranted.

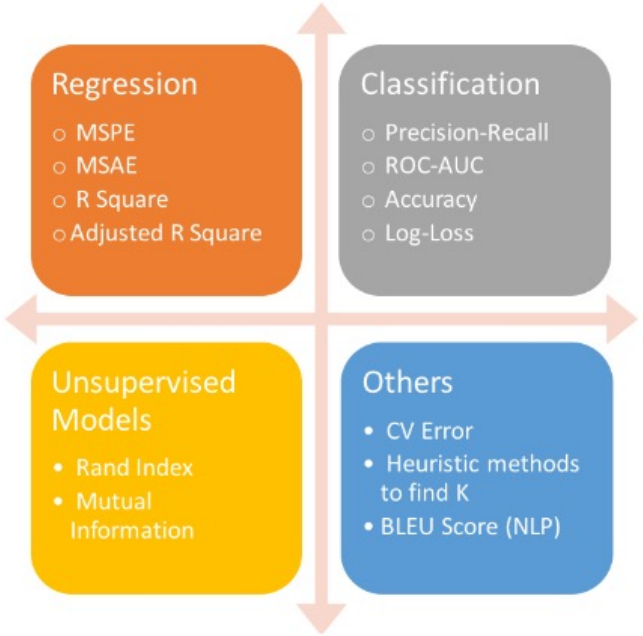
Key Ideas

- Highly imbalanced data (i.e., where the interesting outcomes, the 1s, are rare) are problematic for classification algorithms.
- One strategy for working with imbalanced data is to balance the training data via undersampling the abundant case (or oversampling the rare case).
- If using all the 1s still leaves you with too few 1s, you can bootstrap the rare cases, or use SMOTE to create synthetic data similar to existing rare cases.
- Imbalanced data usually indicates that correctly classifying one class (the 1s) has higher value, and that value ratio should be built into the assessment metric.

평가 지표 정리 (회귀 / 분류)

Summary on Evaluation Index

모델링의 목적	목표 변수 유형	관련 모델	평가 방법
예측 / 회귀 (Prediction)	연속형	선형 회귀	MSE, RMSE, MAE, MAPE 등
분류 (Classification)	범주형	- 로지스틱 회귀 - 의사결정나무 - 서포트벡터머신	정확도, 정밀도, 재현율, F1 -score



- 정밀도, 재현율, 특이도
  - 분류 모형의 목적에 따라 다양한 지표를 볼 수 있음

클래스 ={정상, 불량}		예측한 클래스	
		정상	불량
실제 클래스	정상	TN	FP
	불량	FN	TP

$$\text{정밀도}(Precision) = \frac{\text{옳게 분류된 불량 데이터의 수}}{\text{불량으로 예측한 데이터}} = \frac{TP}{FP+TP}$$

$$\text{재현율}(Recall) = \frac{\text{옳게 분류된 불량 데이터의 수}}{\text{실제 불량 데이터의 수}} = \frac{TP}{FN+TP}$$

$$\text{특이도}(Specificity) = \frac{\text{옳게 분류된 정상 데이터의 수}}{\text{실제 정상 데이터의 수}} = \frac{TN}{TN+FP}$$

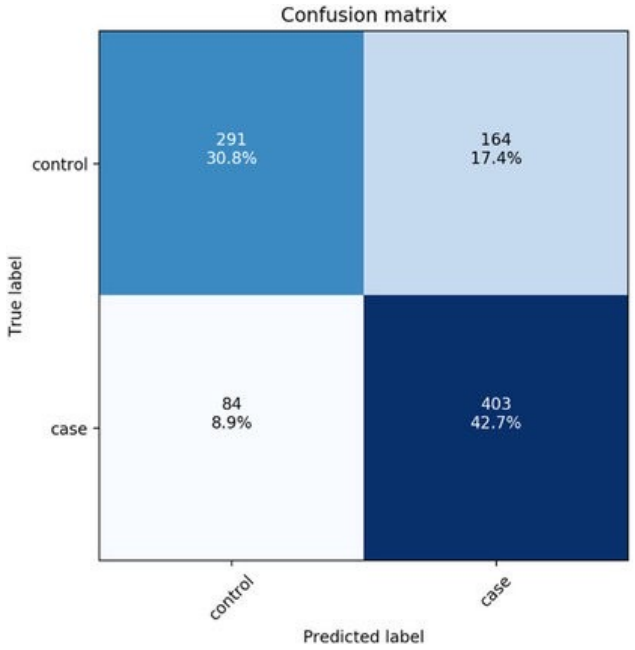
정밀도(precision)는 분류 모형이 불량을 진단하기 위해 얼마나 잘 작동했는지 보여주는 지표

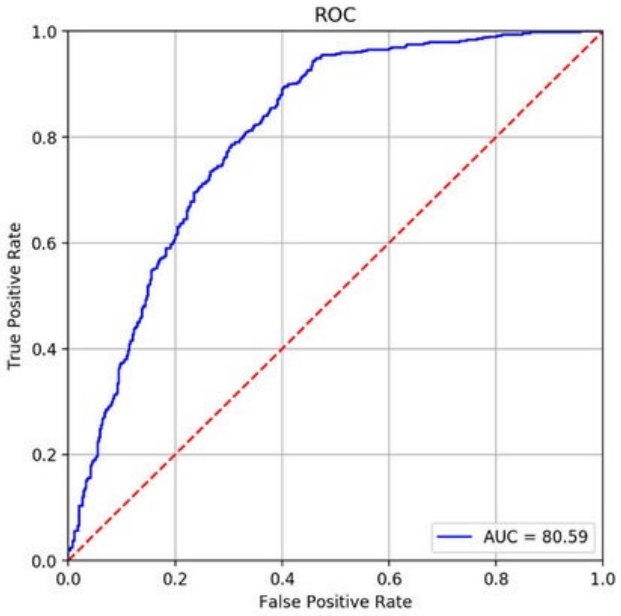
재현율(recall)은 불량 데이터중 실제로 불량이라고 진단한 제품의 비율 (진단 확률)

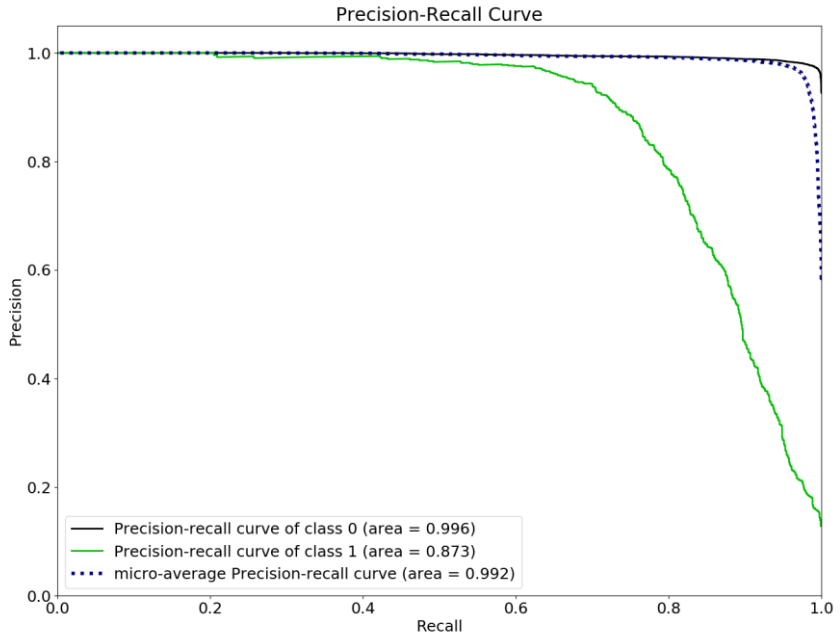
특이도(specificity)는 분류 모형이 정상을 진단하기 위해 잘 작동하는지를 보여주는 지표

평가 지표 정리 (회귀 / 분류)

Summary on Evaluation Index







불균형 데이터셋을 평가하는 경우 (ex. 카드사 사기탐지 모델링)

- ROC AUC에 추가로,
  PR(Precision-Recall) AUC까지 측정하는 것을 권장!!

Q&A and Break Time

질의응답 및 휴식시간 (5분)

2부: 머신러닝 기본

1) 회귀분석 + 예측의 기초

2) SVM (예측 및 분류)

회귀분석 – 선형 회귀

Linear Regression

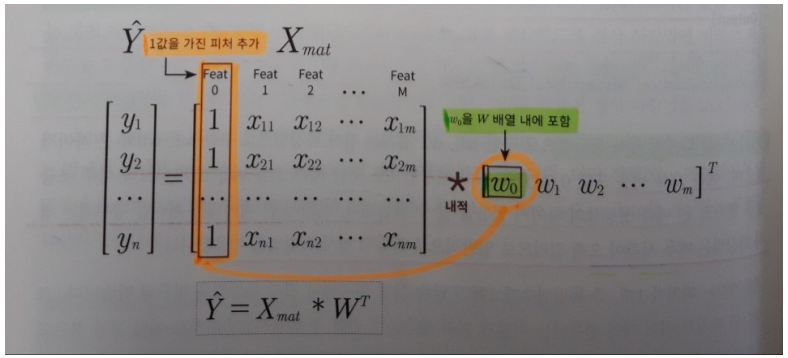
- 다중 선형 회귀분석 모형

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

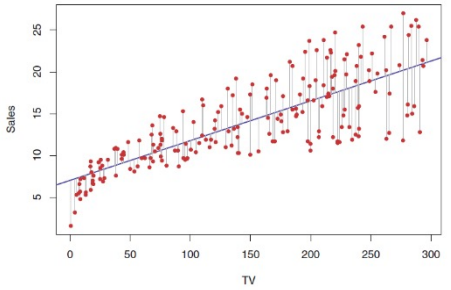
- $\hat{y}$은 예측값입니다.
- n은 특성의 수입니다.
- x_i는 i번째 특성값입니다.
- θ_j는 j번째 모델 파라미터입니다(편향 θ_0과 특성의 가중치 $\theta_1, \theta_2, \cdots, \theta_n$을 포함합니다).

- 벡터 형태의 표현

$$\hat{y} = h_\theta(x) = \theta^T x$$



- 선형 회귀 학습 = RMSE (Root Mean Squared Error)를 최소화 하는 Theta를 찾는 것



$$\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

회귀분석 – 선형 회귀

Linear Regression

- 정규 방정식으로 해답 찾기

$$\hat{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- $\hat{\theta}$은 비용 함수를 최소화하는 θ값입니다.
- $\mathbf{y}$는 $y^{(1)}$부터 $y^{(m)}$까지 포함하는 타깃 벡터입니다.

- 특수 조건
  - m < n이거나 어떤 특성이 중복되어 행렬 $X^T * X$ 의 역행렬이 없다면(즉, 특이 행렬이라면) 정규방정식이 작동하지 않는다. 이러한 경우 유사역행렬을 사용할 수 있다.
  - 정규방정식과 다르게 유사역행렬은 항상 구할 수 있다.
  - 유사역행렬은 특잇값 분해(SVD : Singular Value Decomposition)라는 표준 행렬 분해 기법을 사용해서 계산된다.
  - SVD는 훈련 세트 행렬 X를 3개의 행렬 곱셈 $U * \Sigma * V^T$ 로 분해한다.
  - 따라서 유사역행렬은 $X+ = V * (\Sigma+) * U^T$ 로 계산된다.

회귀분석 – 선형 회귀

Linear Regression

- 정규 방정식으로 해답 찾기

$$\hat{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

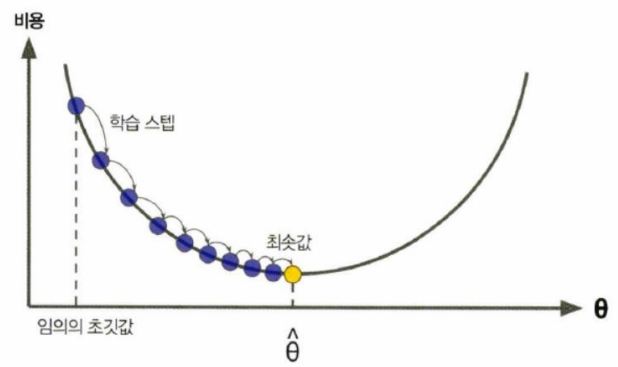
- $\hat{\theta}$은 비용 함수를 최소화하는 θ값입니다.
- $\mathbf{y}$는 $y^{(1)}$부터 $y^{(m)}$까지 포함하는 타깃 벡터입니다.

- 계산 복잡도 (Big-O notation 기반으로)
  - 정규방정식은 계산 복잡도가 일반적으로 $O(n^{2.4})$에서 $O(n^3)$ 사이이다.
  - Scikit Learn의 Linear Regression 클래스가 사용하는 SVD 방법은 약 $O(n^2)$이다.
  - 두 방법 모두 특성 수가 많아지면 매우 느려진다. 이러한 경우 사용하는 방법이 바로 **경사 하강법**이다.

회귀분석 – 선형 회귀

Linear Regression

- 경사 하강법
  - 비용 함수(예시: RMSE)를 최소화하기 위해 반복해서 파라미터를 조정하는 방법

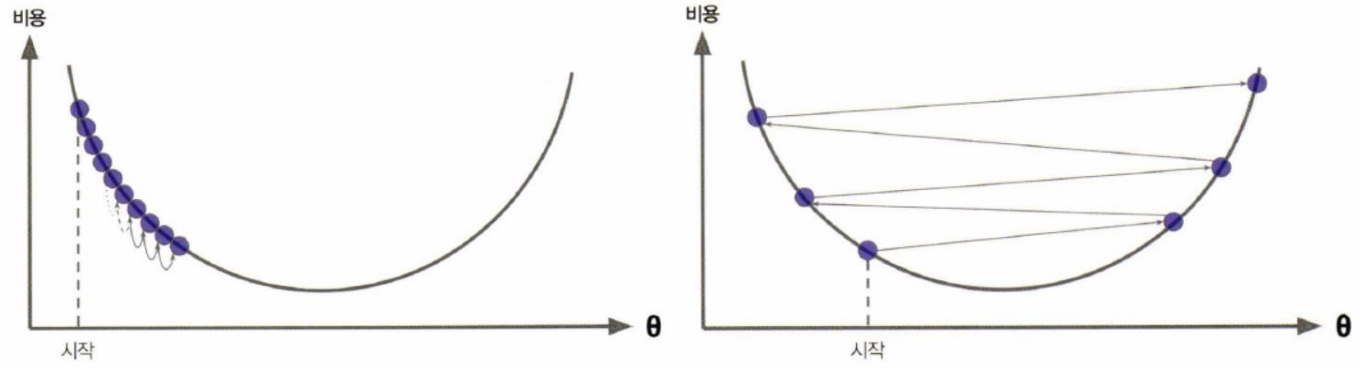


  - θ를 임의의 값으로 시작해서(무작위 초기화) 한 번에 조금씩 비용 함수가 감소되는 방향으로 알고리즘이 최솟값에 수렴할 때까지 점진적으로 향상시킨다.

회귀분석 – 선형 회귀

Linear Regression

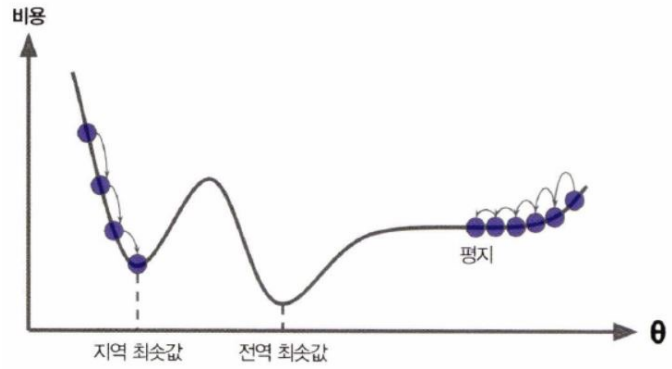
- 경사 하강법의 중요 매개변수
  - 경사 하강법에서 중요한 파라미터는 스텝의 크기, 즉 **학습률(Learning rate)**이다.
  - 학습률(학습 스텝 크기)는 비용 함수의 기울기에 비례하며, 파라미터가 최솟값에 가까워질수록 점차 줄어든다.
  - 학습률이 너무 작으면, 알고리즘이 수렴하기 위해 반복을 많이 진행해야 하므로 시간이 오래 걸린다.
  - 학습률이 너무 크면, 알고리즘을 더 큰 값으로 발산하게 만들어 적절한 해법을 찾지 못할 수 있다.



회귀분석 – 선형 회귀

Linear Regression

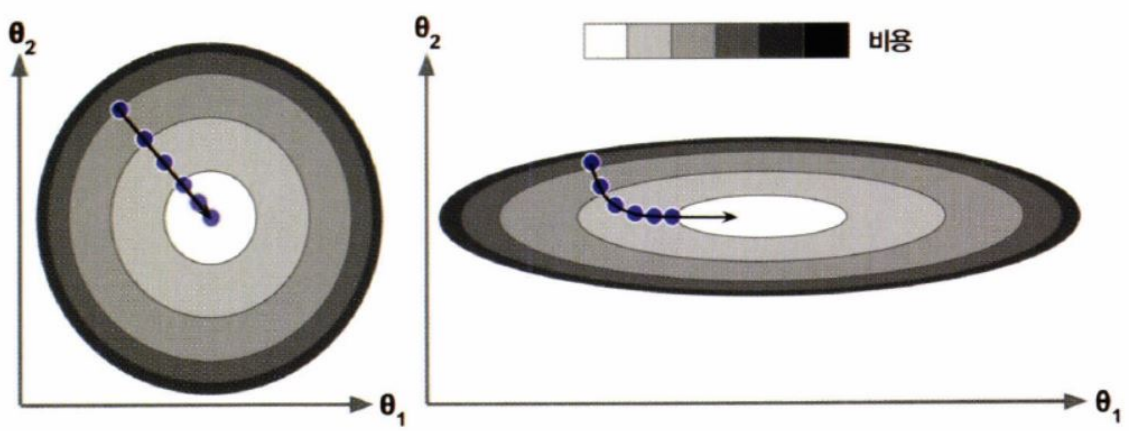
- 경사 하강법의 문제점
  - θ를 임의의 값으로 시작(무작위 초기화)하기 때문에 다음과 같은 문제점들이 발생한다.
  - 알고리즘이 왼쪽에서 시작하면, 전역 최솟값(global minimum)보다 덜 좋은 지역 최솟값(local minimum)에 수렴한다.
  - 반대로 알고리즘이 오른쪽에서 시작하면, 평탄한 지역을 지나기 위해 오랜 시간이 걸리고 일찍 멈추게 되어 전역 최솟값에 도달하지 못한다.



회귀분석 – 선형 회귀

Linear Regression

- 경사 하강법 활용시 유의할 점
  - 반드시 모든 특성이 같은 스케일을 갖도록 만들어야 한다.
  - Scikit-learn의 StandardScaler, MinMaxScaler, RobustScaler 등을 사용할 수 있다.
  - 그렇지 않으면 최솟값으로 수렴하는 데 훨씬 오래 걸린다.



회귀분석 – 선형 회귀

Linear Regression

- 경사 하강법 종류
  - 배치 경사 하강법
  - 확률적 경사 하강법
  - 미니배치 경사 하강법

회귀분석 – 선형 회귀

Linear Regression

- 배치 경사 하강법
  - **전체 데이터 셋에 대한 에러**를 구한 뒤, **기울기를 한 번만 계산**하여 **모델의 파라미터를 업데이트**하는 방법
  - 장점
    - 전체 학습 데이터 셋에 대해 한 번의 업데이트가 이루어지기 때문에, 전체적인 연산 횟수가 적다.
    - 전체 데이터 셋에 대해 Gradient를 계산하여 진행하기 때문에, 최솟값으로의 수렴이 안정적으로 진행된다.
  - 단점
    - 한 스텝에 모든 학습 데이터를 사용하기 때문에 시간이 오래 걸린다.
    - 지역 최적화(local optimal) 상태가 되면 빠져나오기 힘들다.
    - 모델 파라미터의 업데이트가 이루어지기 전까지 모든 학습 데이터 셋에 대해 저장해야 하기 때문에, 많은 메모리가 필요하다.

회귀분석 – 선형 회귀

Linear Regression

- 확률적 경사 하강법
  - 매 스텝에서 **한 개의 샘플을 무작위로 선택**하고, 그 **하나의 샘플에 대한 gradient를 계산**하는 방법
  - 장점
    - 배치 경사 하강법보다 알고리즘이 훨씬 빠르다.
    - 메모리 차지 비중이 적다.
  - 단점
    - 비용 함수가 최솟값에 도달할 때까지 위아래로 요동치면서 평균적으로 감소하기 때문에, 배치 경사 하강법보다 훨씬 불안정하다.
    - 따라서 알고리즘이 멈출 때, 좋은 파라미터가 구해지겠지만 최적치는 아니다.

회귀분석 – 선형 회귀

Linear Regression

- 미니배치 경사 하강법
  - **미니배치**라 부르는 **임의의 작은 샘플 세트**에 대해 gradient를 계산하는 방법
  - 장점
    - 행렬 연산에 최적화된 하드웨어, 특히 GPU를 사용해서 향상된 성능을 보인다.
    - 미니배치를 어느 정도 크게 하면, 알고리즘은 파라미터 공간에서 SGD보다 덜 불규칙하게 움직인다.
    - 즉, 미치배치 경사 하강법이 SGD보다 최솟값에 더 가까이 도달하게 될 것이다.
  - 단점
    - SGD보다 지역 최솟값(local minimum)에서 빠져나오기는 더 힘들 것이다.

회귀분석 – 선형 회귀

Linear Regression

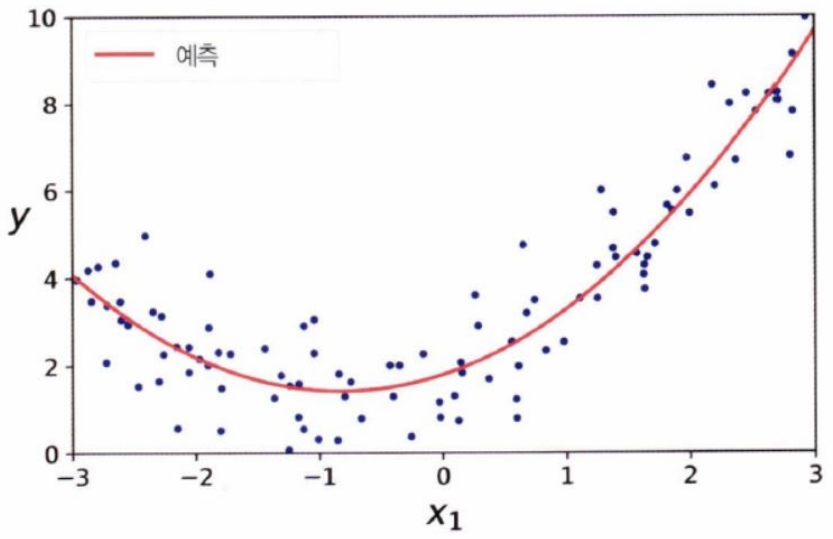
- 선형 회귀 최적화 방법 요약

알고리즘	m(샘플 수)이 클 때	외부 메모리 학습 지원	n(특성 수)이 클 때	하이퍼 파라미터 수	스케일 조정 필요 여부	사이킷런
정규방정식	빠름	x	느림	0	x	N/A
SVD(특이값 분해)	빠름	x	느림	0	x	LinearRegression
배치 경사 하강법	느림	x	빠름	2	o	SGDRegressor
확률적 경사 하강법	빠름	o	빠름	>= 2	o	SGDRegressor
미니배치 경사 하강법	빠름	o	빠름	>= 2	o	SGDRegressor

회귀분석 – 다중 회귀

Polynomial Regression

- 다중 회귀
  - 통계에서는 비선형, 하지만 머신러닝에서는 선형으로 분류! (Why? How?)
    - 비선형 데이터를 학습하는 데 선형 모델을 사용할 수 있기 때문 (Scikit-learn의 PolynomialFeatures 활용)
  - 각 특성의 거듭제곱을 새로운 특성으로 추가하고, 추가된 특성을 포함한 데이터 셋에 선형 모델을 훈련시키는 것이다.
    - 변수 a, b의 3차원(degree=3) polynomial feature 생성 예시
      - a^2, b^2, a^3, b^3 뿐만 아니라 ab, a^2b, ab^2도 특성으로 포함된다.



회귀분석 – 학습 곡선

Learning Curve

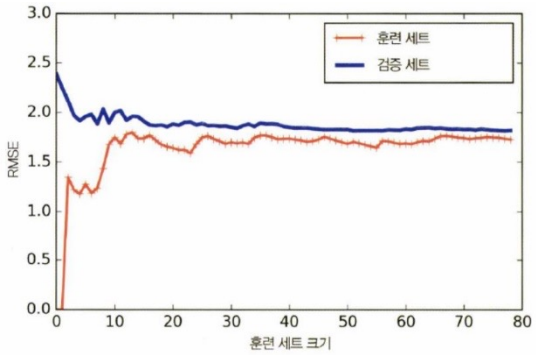
- 학습 곡선
  - 학습 곡선은 모델의 일반화 성능을 추정하기 위한 방법 중 하나이다.
  - 이 그래프는 훈련 세트와 검증 세트의 모델 성능을 훈련 세트 크기의 함수로 나타낸다.

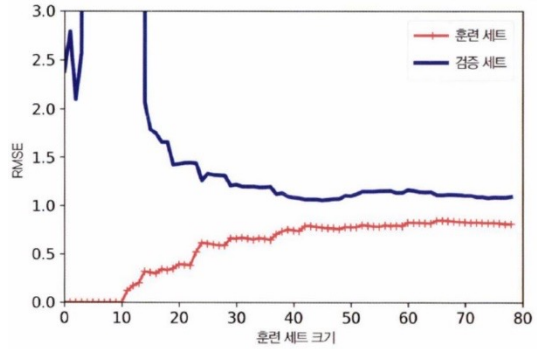
```python
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def plot_learning_curves(model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train[:m], y_train_predict))
        val_errors.append(mean_squared_error(y_val, y_val_predict))
    plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="훈련 세트") #RMSE 측정
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="검증 세트")
```

회귀분석 – 학습 곡선

Learning Curve

- 선형 회귀 모델의 학습 곡선
  - 두 곡선이 수평한 구간을 만들고 꽤 높은 오차에서 매우 가까이 근접해있다.
  - 이는 과소적합 모델의 전형적인 학습 곡선 모습이다.
    (학습 및 예측 능력이 높지 않음)
- 다중 회귀 모델의 학습 곡선
  - 훈련 데이터의 오차가 앞서 확인한 단순 선형 회귀 모델보다 훨씬 낮다.
  - 또한 두 곡선 사이에 공간이 존재하는데, 훈련 데이터에서의 모델 성능이 검증 데이터에서보다 훨씬 낫다는 것을 의미한다. 즉, 과대적합되었다는 말이다.
  - 하지만 더 큰 훈련 세트를 사용하면, 두 곡선 사이의 공간이 점점 가까워질 것이다.
  - 이처럼 과대적합 모델을 개선하는 한 가지 방법은, 검증 오차가 훈련 오차에 근접할 때까지 더 많은 훈련 데이터를 추가하는 것이다.

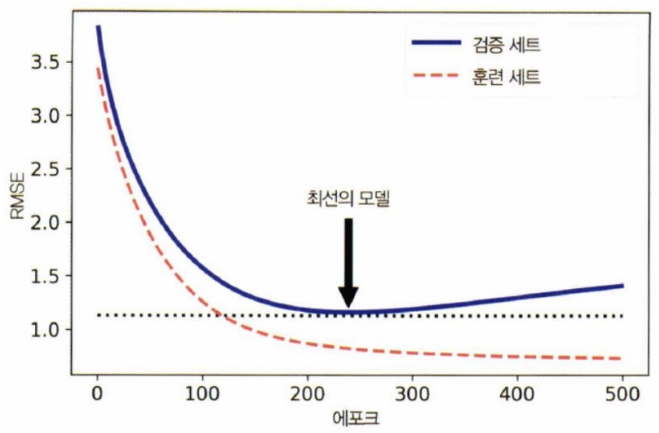




회귀분석 – 조기 종료

Early Stopping

- 조기 종료 (Early Stopping)
  - 경사 하강법과 같은 반복적인 학습 알고리즘을 규제하는 방식
    (검증 에러가 최솟값에 도달하면 바로 훈련을 중지시키는 것)
  - **매우 효과적이고 간단**



회귀분석 – 로지스틱 회귀

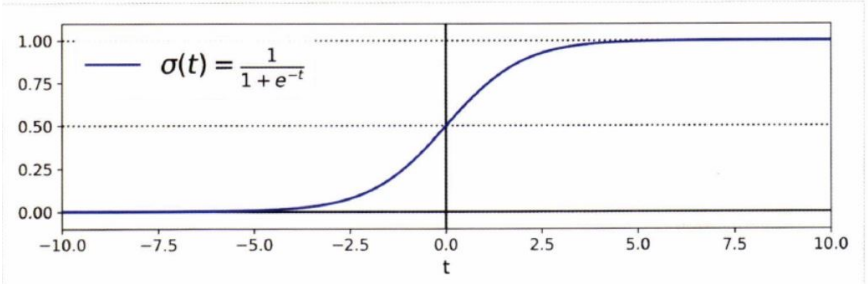
Logistic Regression

- 로지스틱 회귀 (Logistic Regression)
  - **Target 변수**가 **범주형 (Not 수치형)**인 경우 사용하는 모델이다.
  - 샘플이 특정 클래스에 속할 **확률을 추정**하는데 널리 사용된다.
    - 각 클래스에 속할 확률을 비교해서, 확률 값이 큰 쪽으로 분류한다.
    - 일반적으로 **분류 기준 확률 값(Threshold)**으로 0.5를 사용한다.
      - 이를 이진 분류기(Binary Classifier)라고 한다.

$$\hat{y} = \begin{cases} 0 & \hat{p} < 0.5\text{일 때} \\ 1 & \hat{p} \geq 0.5\text{일 때} \end{cases}$$

  - 로지스틱 회귀는 로지스틱 함수(시그모이드 함수)를 사용하여 확률 값을 추정하며, 출력 결과 값은 0과 1사이의 값을 갖는다.

$$\sigma(t) = \frac{1}{1+\exp(-t)}$$



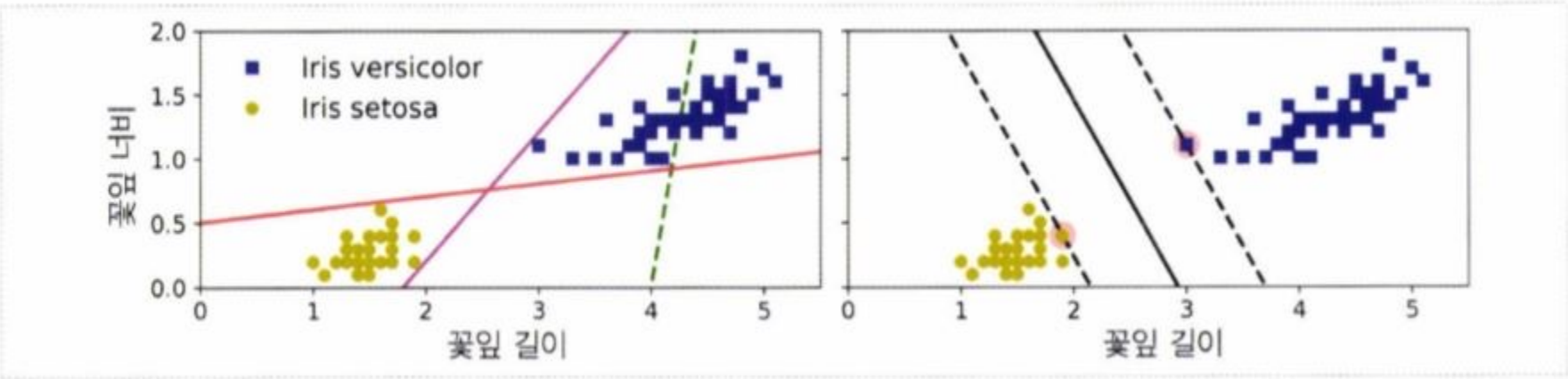
SVM (Support Vector Machine)

- 특징
  - 선형/비선형 기반 분류, 회귀, 이상치 탐색에 활용 가능한 모델
  - 가장 인기있는 모델에 속함
  - 반드시 알고 있어야 하는 모델
  - 복잡한 분류 문제에 잘 들어맞음
  - 작거나 중간 크기의 데이터셋에 적합
- 내용
  - 선형 SVM 분류
  - 비선형 SVM 분류
  - SVM 회귀
  - SVM 이론

SVM – 선형 SVM 분류

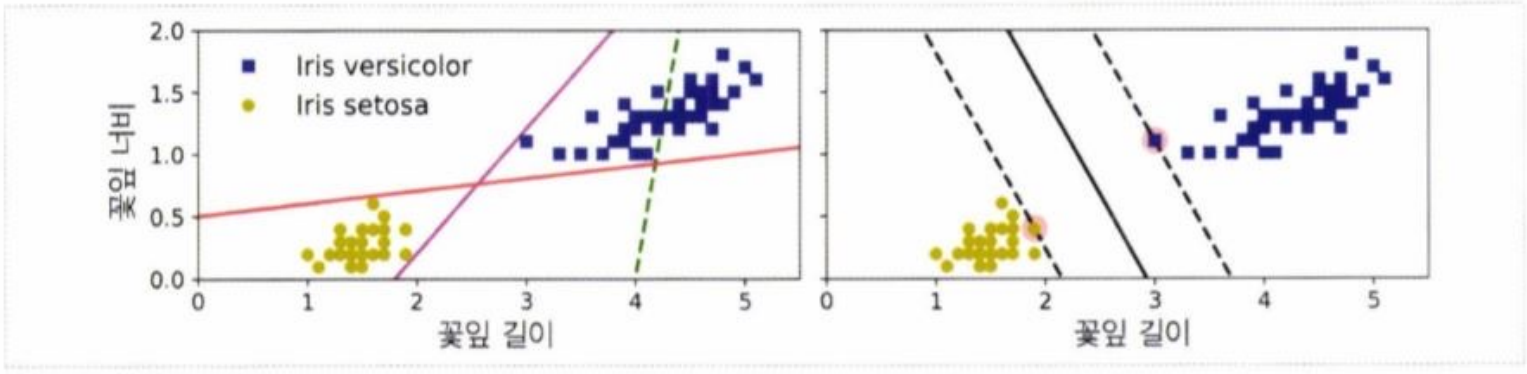
선형 SVM 분류

- SVM 분류기(또는 Large Margin Classification)는 클래스 사이에 폭이 가장 넓은 도로를 찾는 것이라고 생각하면 이해하기 쉽다.
- 마진(Margin)이란?
  - 두 데이터 군과 결정 경계와 떨어져 있는 정도
- 목표: **margin을 최대화하는 결정 경계(선 또는 면)를 찾아내는 것!!**



SVM – 선형 SVM 분류

선형 SVM 분류



- 위 그림을 보면 알 수 있듯이, 도로 바깥쪽에 훈련 샘플을 추가해도 결정 경계에는 영향을 미치지 않는다.
- 또한 결정 경계는 도로 경계에 위치한 샘플. 즉, 서포트 벡터에 의해 결정된다.
  - 서포트 벡터란?
    - 두 클래스 사이의 경계에 위치한 데이터 포인트들
    - 결정 경계를 만드는 데 영향을 미치기 때문에 서포트 벡터라고 부른다.

SVM – 선형 SVM 분류

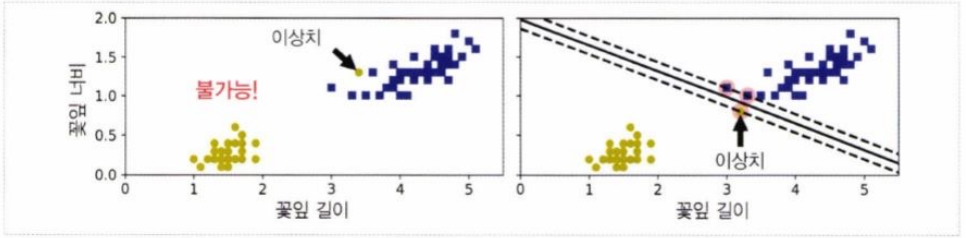
SVM 특징

- 특성의 스케일에 민감하다.
  - 따라서 StandardScaler( )와 같은 스케일링 작업을 수행해주어야 훨씬 더 좋은 결정 경계를 얻을 수 있다.
- 선형 SVM 분류기를 훈련시킬 때, 일반적인 확률적 경사 하강법을 적용한다.
  - 비록 속도는 LinearSVC보다 느리지만, 데이터 셋이 너무 크거나 온라인 학습으로 분류 문제를 다룰 때 유용하다.

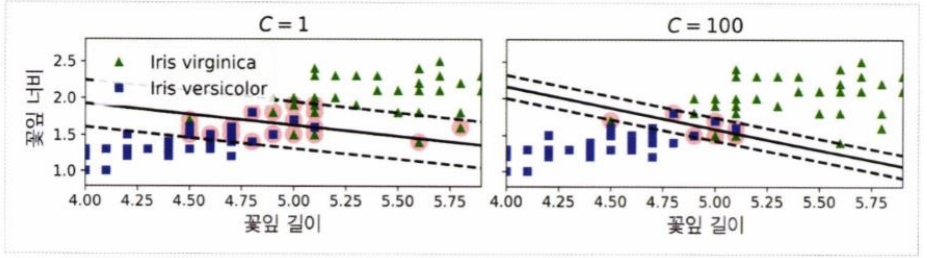
SVM – 선형 SVM 분류

하드 마진 분류 / 소프트 마진 분류

- 하드 마진 분류
  - **모든 샘플이 도로 바깥 쪽에** 올바르게 분류하는 분류기
  - 단점
    - 데이터가 선형적으로 구분될 수 있어야 함
    - 이상치에 민감



- 소프트 마진 분류
  - 도로 폭을 최대한 넓게 유지하는 것과 마진 오류 사이에 적절한 균형을 이루는 분류기
  - Scikit-learn에서 SVM 모델을 만들 때, C라는 하이퍼파라미터를 통해 마진 오류를 조정할 수 있다.
    - **하이퍼파라미터 C**
      - SVM 모델이 과대적합이라면, C를 감소시켜서 모델을 규제할 수 있다.
      - **얼마나 많은 데이터 샘플이 다른 클래스에 놓이는 것을 허용하는지 결정**
      - C가 작을수록 많이 허용, C가 클수록 적게 허용



SVM – 선형 SVM 분류

참고사항

- SVM 분류기는 로지스틱 회귀 분류기와 다르게, 클래스에 대한 확률을 제공하지 않는다.
  - Scikit-learn의 LinearSVC는 predict_proba( ) 메소드를 제공하지 않지만,
    SVC 모델은 probability = True로 매개변수를 지정하면 predict_proba( ) 메소드를 제공한다.
  - SVC 모델의 probability 매개변수 default 값은 False이다.

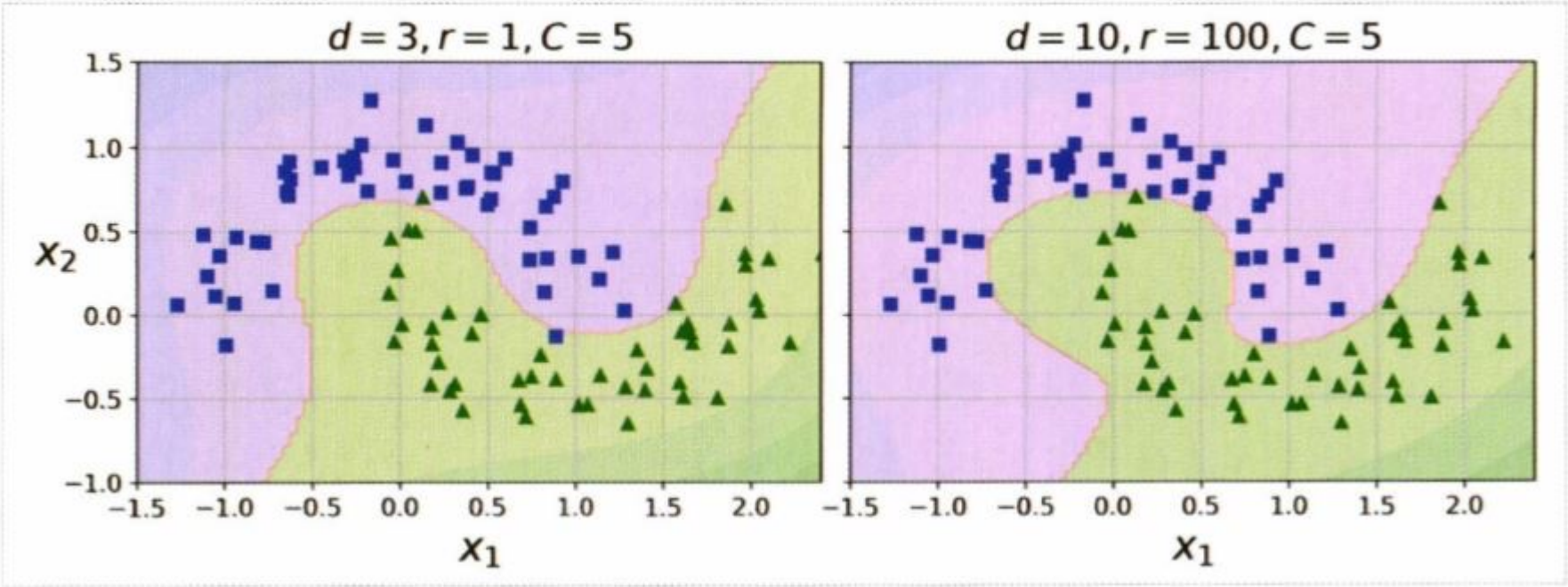
SVM – 비선형 SVM 분류

Polynomial 특성 추가 VS 커널 트릭

- Polynomial 특성 추가 방법론의 단점
  - **낮은 차수의 다항식은 과소적합**을 발생시키고, **높은 차수의 다항식은 과대적합**을 발생시키는 문제점이 있다.
- 커널트릭
  - 이러한 문제점을 보완하기 위해 사용하는 것이 바로 **커널 트릭**이다.
  - 다항식 커널 활용시
    - 실제로는 특성을 추가하지 않으면서, 다항식 특성을 많이 추가한 것과 동일한 결과를 얻을 수 있다.
    - 실제로 어떠한 특성도 추가하지 않으므로 수많은 특성 조합이 생기지 않는다.
    - 즉, 수많은 특성 조합을 생성하지 않으므로 모델이 느려지는 것을 방지할 수 있다.

SVM – 비선형 SVM 분류

커널 트릭

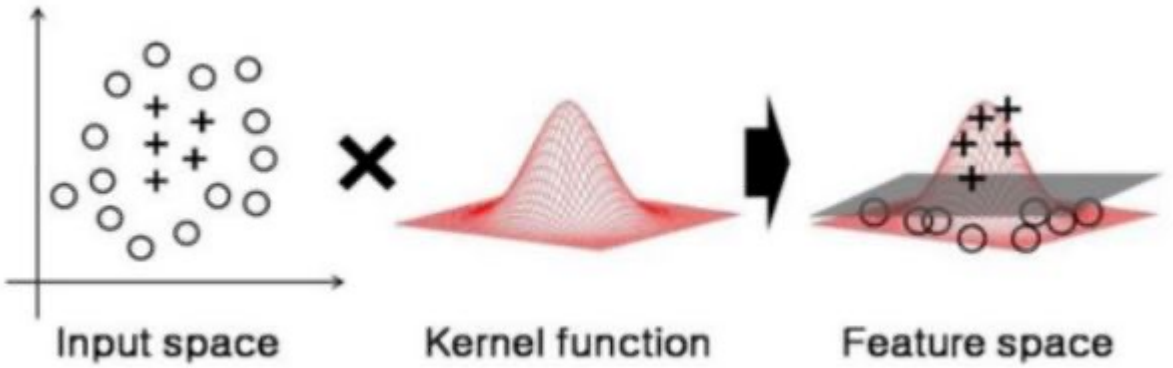


3차 다항식 커널을 사용한 SVM 분류기(왼쪽), 10차 다항식 커널을 사용한 SVM 분류기(오른쪽)

SVM – 비선형 SVM 분류

유사도 특성

- 비선형 분류를 하기 위해서 주어진 데이터를 고차원 특징 공간으로 확장시키는 작업이 필요하다.
  - 이를 커널 기법이라고 한다.



  - 다양한 유사도 함수 중, 가장 많이 사용되는 것은 가우시안 방사 기저 함수(RBF)이다.

SVM – 비선형 SVM 분류

TIP

- 모델의 복잡도를 조절하려면, gamma와 C를 함께 조정해주는 것이 좋다.
- 여러 가지 커널 중, 가장 먼저 선형 커널을 시도해보는 것이 좋다.
- 선형 커널이 훨씬 더 빠르기 때문이다.
- 특히 훈련 세트가 매우 크거나, 특성 수가 많을 때!!
- 훈련 세트가 너무 크지 않으면, 가우시안 RBF 커널도 시도해 볼만하다.

SVM 분류

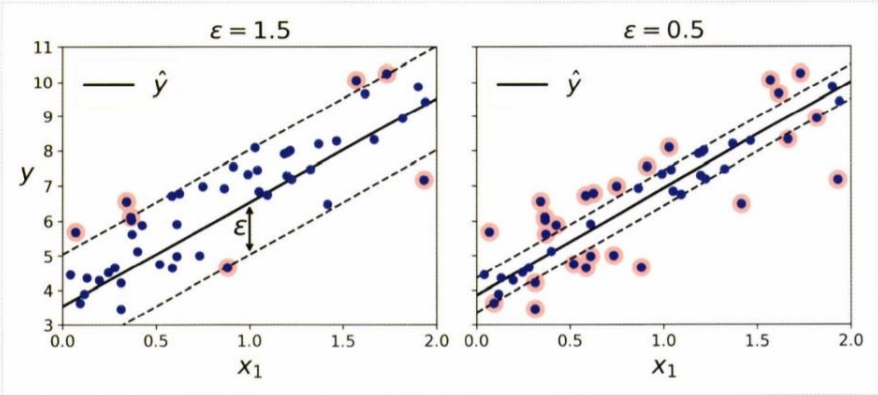
SVM 분류 내용 정리

파이썬 클래스	시간 복잡도	외부 메모리 학습 지원	스케일 조정의 필요성	커널 트릭
LinearSVC	O(m x n)	X	O	X
SGDClassifier	O(m x n)	O	O	X
SVC	O(m^2 x n) ~ O(m^3 x n)	X	O	O

SVM – SVM 회귀

SVM 회귀

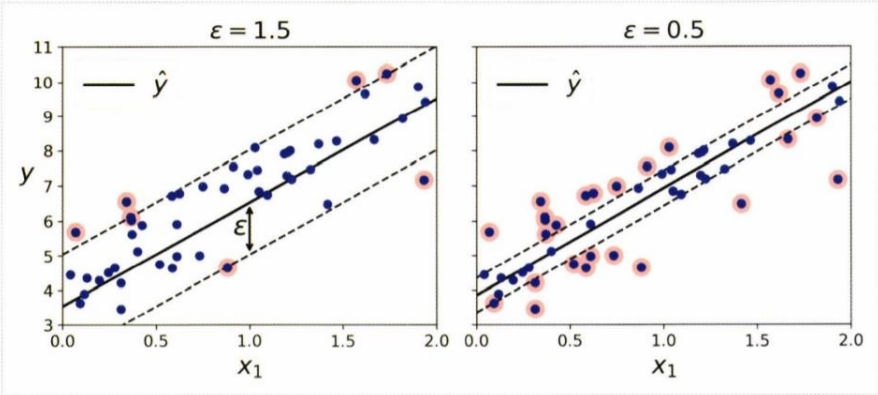
- 일정한 마진 오류 안에서 두 클래스 간의 도로 폭이 가능한 최대가 되도록 하는 것은 SVM 분류와 동일하다.
- 차이점은 SVM 회귀는 **제한된 마진 오류(도로 밖의 샘플) 안에서 도로 안에 가능한 한 많은 샘플이 들어가도록 학습한다**는 점이다.
- 도로 폭은 **epsilon**으로 조절한다.
- SVM을 **분류가 아닌 회귀에 적용하려면, 목표를 반대로 설정**하면 된다.



SVM – SVM 회귀

SVM 회귀 주의할 점

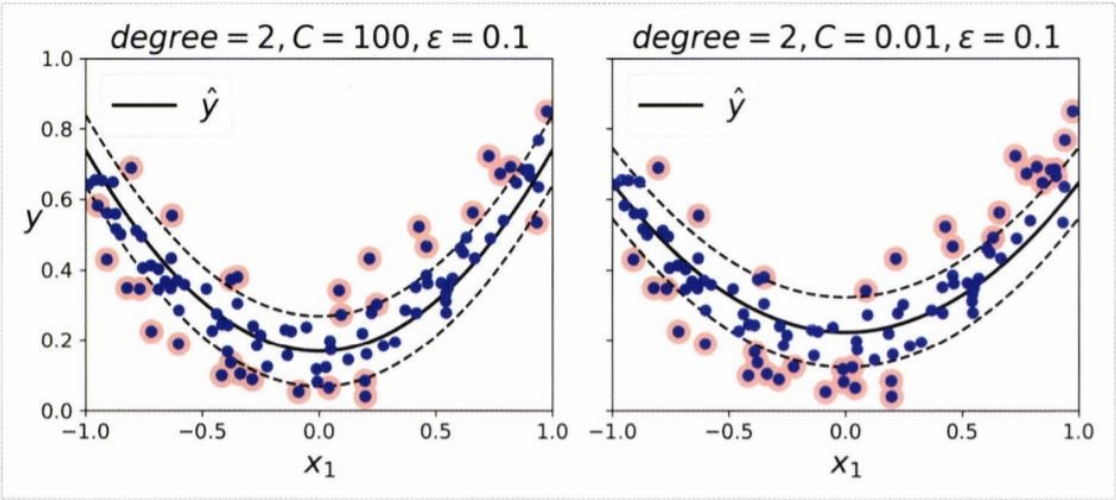
- tolerance(허용오차)와 epsilon(도로의 폭)은 다른 것이다!!
  - 허용오차는 tol 매개변수, 도로의 폭은 epsilon 매개변수로 지정한다.
- 마진 안에서는 훈련 샘플이 추가되어도 모델의 예측에는 영향이 주지 않는다.
  - 즉, epsilon에 민감하지 않은 모델이다.



SVM – 비선형 SVM 회귀

SVM 비선형 회귀

- 비선형 회귀 작업을 처리해주려면, 커널 SVM 모델을 사용해야 한다.
- 이해하기 쉽도록, 아래에 2차 다항 커널을 사용한 SVM 회귀 예시 그림을 첨부하였다.



(왼쪽)규제가 거의 없는 경우(즉, 아주 큰 C)와 (오른쪽)규제가 많은 경우(즉, 아주 작은 C)

Q&A

질의응답 (10분)

끝. 감사합니다.

수업 듣느라 수고하셨습니다.