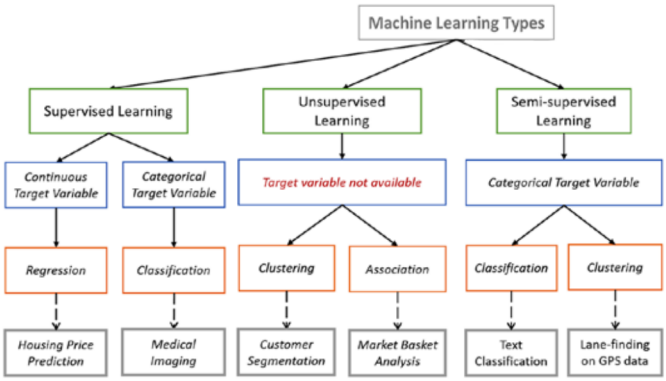
Machine Learning 101

Aug 19, 2019

Contents


Machine Learning Types
Supervised Learning
Unsupervised Learning
Semi-supervised Learning
Continuous Target Variable
Categorical Target Variable
Target variable not available
Categorical Target Variable
Regression
Classification
Clustering
Association
Classification
Clustering
Housing Price Prediction
Medical Imaging
Customer Segmentation
Market Basket Analysis
Text Classification
Lane-finding on GPS data

CHAPTER 1

Introduction

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to learn (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. Machine learning is the design and study of software artifacts that use past experience to make future decisions. It is the study of programs that learn from data. The fundamental goal of machine learning is to generalize, or to induce an unknown rule from examples of the rule's application. Machine learning systems are often described as learning from experience either with or without supervision from humans.

The Basic three different learning styles in machine learning algorithms:

1.1 Supervised Learning

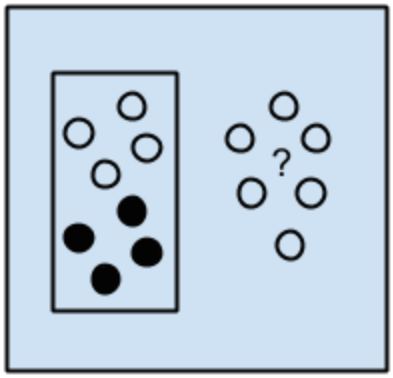
A model is prepared through a training process in which it is required to make predictions and is corrected when those predictions are wrong. The training process continues until the model achieves a desired level of accuracy on the training data. A program predicts an output for an input by learning from pairs of labeled inputs and outputs, the program learns from examples of the right answers.

1. Input data is called training data and has a known label or result such as spam/not-spam or a stock price at a time.
2. Example problems are classification and regression.
3. Example algorithms include Logistic Regression and the Back Propagation Neural Network.

1.2 Unsupervised Learning

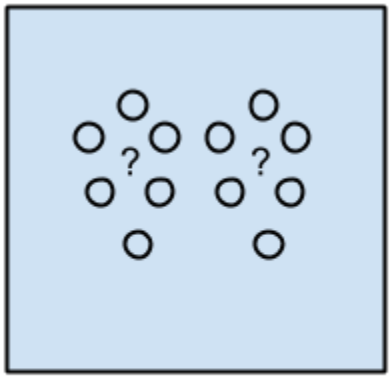
A model is prepared by deducing structures present in the input data. This may be to extract general rules. It may be through a mathematical process to systematically reduce redundancy, or it may be to organize data by similarity. A program does not learn from labeled data. Instead, it attempts to discover patterns in the data.

1. Input data is not labeled and does not have a known result.
2. Example problems are clustering, dimensionality reduction and association rule learning.



Supervised Learning
Algorithms

3. Example algorithms include: the Apriori algorithm and k-Means.

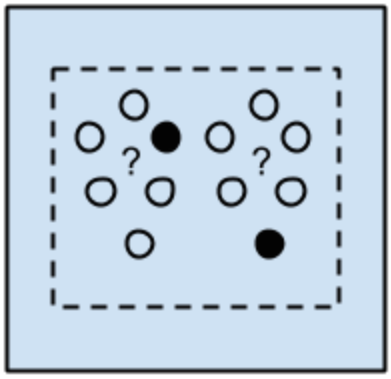


Unsupervised Learning
Algorithms

1.3 Semi-Supervised Learning

There is a desired prediction problem but the model must learn the structures to organize the data as well as make predictions. Semi-supervised learning problems, make use of both supervised and unsupervised data; these problems are located on the spectrum between supervised and unsupervised learning

1. Input data is a mixture of labeled and unlabeled examples.
2. Example problems are classification and regression.
3. Example algorithms are extensions to other flexible methods that make assumptions about how to model the unlabeled data.



Semi-supervised
Learning Algorithms

1.4 Key Terms

1.4.1 Model

A machine learning model can be a mathematical representation of a real-world process. To generate a machine learning model you will need to provide training data to a machine learning algorithm to learn from.

1.4.2 Algorithm

Machine Learning algorithm is the hypothesis set that is taken at the beginning before the training starts with real-world data. When we say Linear Regression algorithm, it means a set of functions that define similar characteristics as defined by Linear Regression and from those set of functions we will choose one function that fits the most by the training data.

1.4.3 Training

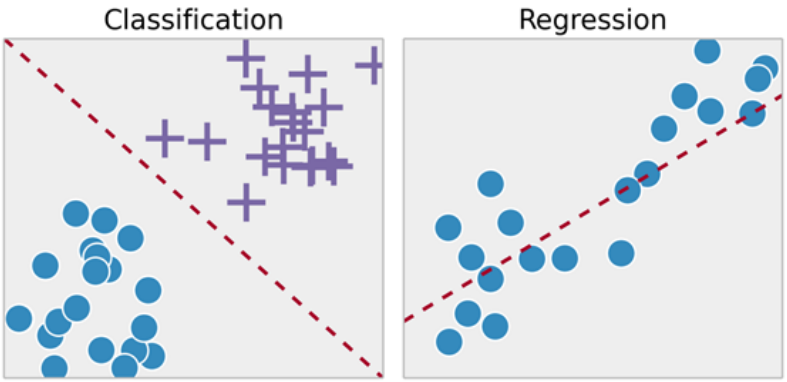
While training for machine learning, you pass an algorithm with training data. The learning algorithm finds patterns in the training data such that the input parameters correspond to the target. The output of the training process is a machine learning model which you can then use to make predictions. This process is also called "learning".

1.4.4 Regression

Regression techniques are used when the output is real-valued based on continuous variables. For example, any time series data. This technique involves fitting a line.

1.4.5 Classification

In classification, you will need to categorize data into predefined classes. For example, an email can either be 'spam' or 'not spam'.



1.4.6 Target

The target is whatever the output of the input variables. It could be the individual classes that the input variables maybe mapped to in case of a classification problem or the output value range in a regression problem. If the training set is considered then the target is the training output values that will be considered.

1.4.7 Feature

Features are individual independent variables that act as the input in your system. Prediction models use features to make predictions. New features can also be obtained from old features using a method known as 'feature engineering'. More simply, you can consider one column of your data set to be one feature. Sometimes these are also called attributes. And the number of features are called dimensions.

1.4.8 Label

Labels are the final output. You can also consider the output classes to be the labels. When data scientists speak of labeled data, they mean groups of samples that have been tagged to one or more labels.

1.4.9 Overfitting

An important consideration in machine learning is how well the approximation of the target function that has been trained using training data, generalizes to new data. Generalization works best if the signal or the sample that is used as the training data has a high signal to noise ratio. If that is not the case, generalization would be poor and we will not get good predictions. A model is overfitting if it fits the training data too well and there is a poor generalization of new data.

1.4.10 Regularization

Regularization is the method to estimate a preferred complexity of the machine learning model so that the model generalizes and the over-fit/under-fit problem is avoided. This is done by adding a penalty on the different parameters of the model thereby reducing the freedom of the model.

1.4.11 Parameter and Hyper-Parameter

Parameters are configuration variables that can be thought to be internal to the model as they can be estimated from the training data. Algorithms have mechanisms to optimize parameters. On the other hand, hyperparameters cannot be estimated from the training data. Hyperparameters of a model are set and tuned depending on a combination of some heuristics.

Citations

References

1. Machine Learning Types

CHAPTER 2

Learning Models

The process of training an ML model involves providing an ML algorithm (that is, the learning algorithm) with training data to learn from. The term ML model refers to the model artifact that is created by the training process.

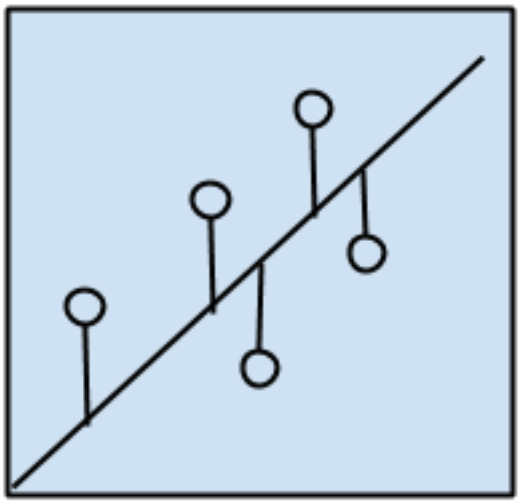
The training data must contain the correct answer, which is known as a target or target attribute. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict), and it outputs an ML model that captures these patterns. You can then use the ML model to get predictions on new data for which you do not know the target.

Organizing machine learning algorithms is useful because it forces you to think about the roles of the input data and the model preparation process and select one that is the most appropriate for your problem in order to get the best result. Algorithms are often grouped by similarity in terms of their function.

2.1 Regression Algorithms

Regression is concerned with modeling the relationship between variables that is iteratively refined using a measure of error in the predictions made by the model. Regression methods are a workhorse of statistics and have been co-opted into statistical machine learning. This may be confusing because we can use regression to refer to the class of problem and the class of algorithm. Really, regression is a process.

1. Ordinary Least Squares Regression (OLSR)
2. Linear Regression
3. Logistic Regression
4. Stepwise Regression
5. Multivariate Adaptive Regression Splines (MARS)
6. Locally Estimated Scatterplot Smoothing (LOESS)



Regression Algorithms

2.2 Instance-based Algorithms

Instance-based learning model is a decision problem with instances or examples of training data that are deemed important or required to the model. Such methods typically build up a database of example data and compare new data to the database using a similarity measure in order to find the best match and make a prediction. For this reason, instance-based methods are also called winner-take-all methods and memory-based learning. Focus is put on the representation of the stored instances and similarity measures used between instances.

1. k-Nearest Neighbor (kNN)
2. Learning Vector Quantization (LVQ)
3. Self-Organizing Map (SOM)
4. Locally Weighted Learning (LWL)

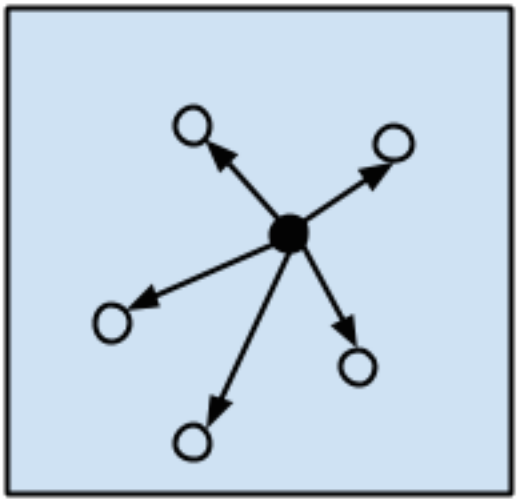
2.3 Regularization Algorithms

An extension made to another method (typically regression methods) that penalizes models based on their complexity, favoring simpler models that are also better at generalizing. regularization algorithms

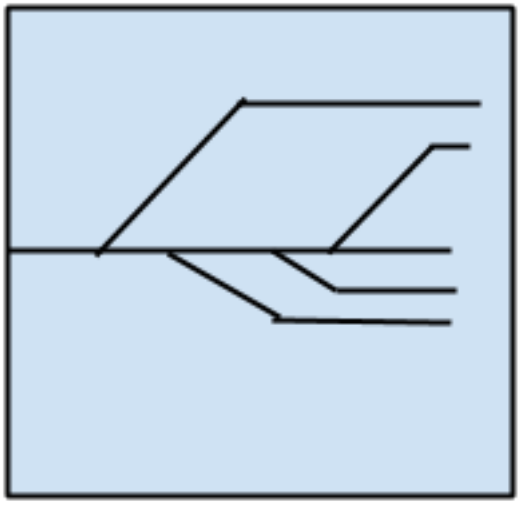
1. Ridge Regression
2. Least Absolute Shrinkage and Selection Operator (LASSO)
3. Elastic Net
4. Least-Angle Regression (LARS)

2.4 Decision Tree Algorithms

Decision tree methods construct a model of decisions made based on actual values of attributes in the data. Decisions fork in tree structures until a prediction decision is made for a given record. Decision trees are trained on data



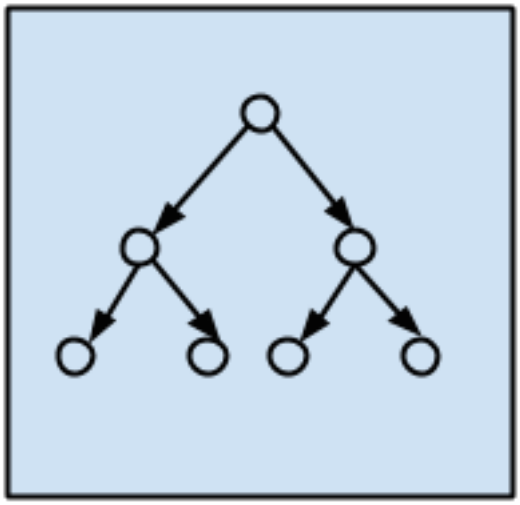
Instance-based
Algorithms



Regularization
Algorithms

for classification and regression problems. Decision trees are often fast and accurate and a big favorite in machine learning.

1. Classification and Regression Tree (CART)
2. Iterative Dichotomiser 3 (ID3)
3. C4.5 and C5.0 (different versions of a powerful approach)
4. Chi-squared Automatic Interaction Detection (CHAID)
5. Decision Stump
6. M5
7. Conditional Decision Trees

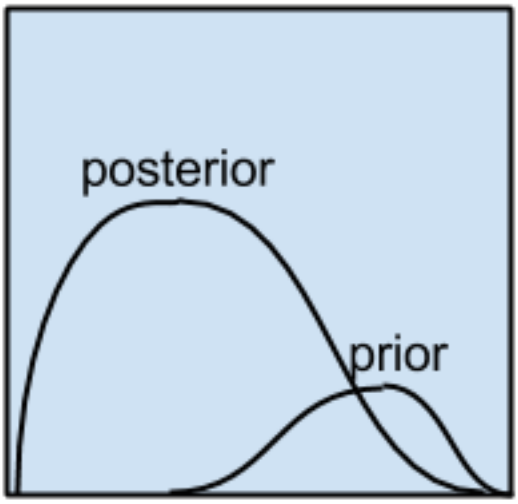


Decision Tree Algorithms

2.5 Bayesian Algorithms

Bayesian methods are those that explicitly apply Bayes' Theorem for problems such as classification and regression. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability of the response variable belonging to a particular value.

1. Naive Bayes
2. Gaussian Naive Bayes
3. Multinomial Naive Bayes
4. Averaged One-Dependence Estimators (AODE)
5. Bayesian Belief Network (BBN)
6. Bayesian Network (BN)

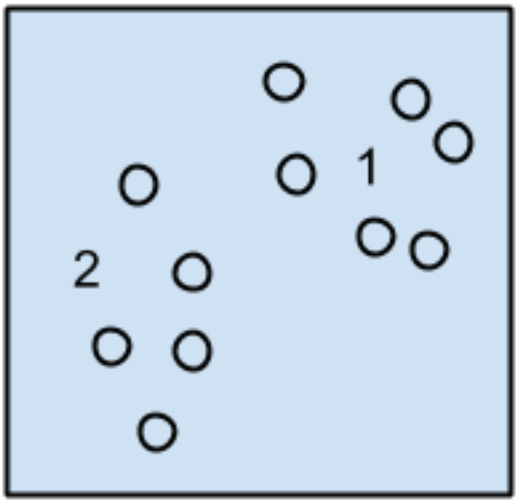


Bayesian Algorithms

2.6 Clustering Algorithms

Clustering, like regression, describes the class of problem and the class of methods. Clustering methods are typically organized by the modeling approaches such as centroid-based and hierarchal. All methods are concerned with using the inherent structures in the data to best organize the data into groups of maximum commonality.

1. k-Means
2. k-Medians
3. Expectation Maximisation (EM)
4. Hierarchical Clustering

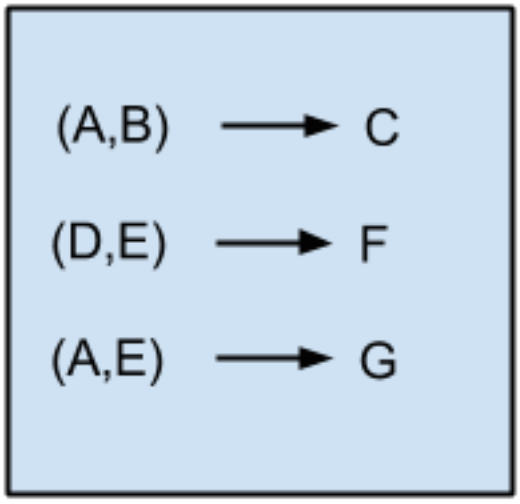


Clustering Algorithms

2.7 Association Rule Learning Algorithms

Association rule learning methods extract rules that best explain observed relationships between variables in data. These rules can discover important and commercially useful associations in large multidimensional datasets that can be exploited by an organization.

1. Apriori algorithm
2. Eclat algorithm



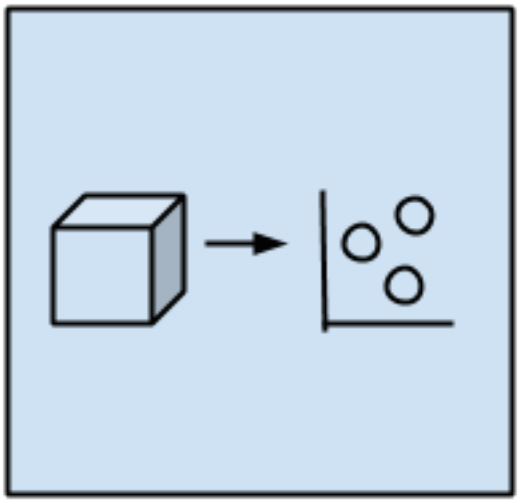
Association Rule
Learning Algorithms

2.8 Dimensionality Reduction Algorithms

Like clustering methods, dimensionality reduction seek and exploit the inherent structure in the data, but in this case in an unsupervised manner or order to summarize or describe data using less information. This can be useful to visualize dimensional data or to simplify data which can then be used in a supervised learning method. Many of these methods can be adapted for use in classification and regression.

1. Principal Component Analysis (PCA)
2. Principal Component Regression (PCR)
3. Partial Least Squares Regression (PLSR)
4. Sammon Mapping
5. Multidimensional Scaling (MDS)
6. Projection Pursuit
7. Linear Discriminant Analysis (LDA)
8. Mixture Discriminant Analysis (MDA)
9. Quadratic Discriminant Analysis (QDA)

10. Flexible Discriminant Analysis (FDA)



2.9 Ensemble Algorithms

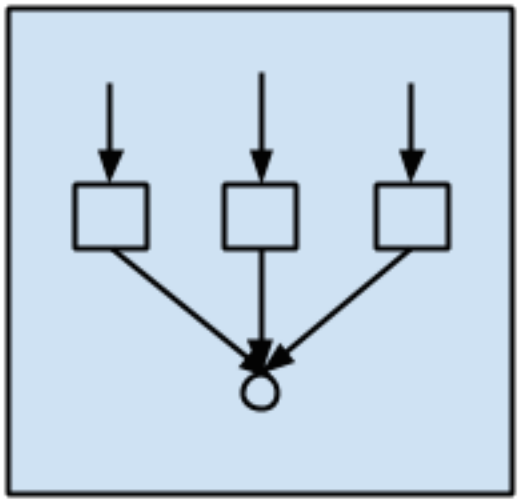
Ensemble methods are models composed of multiple weaker models that are independently trained and whose predictions are combined in some way to make the overall prediction. Much effort is put into what types of weak learners to combine and the ways in which to combine them. This is a very powerful class of techniques and as such is very popular.

1. Boosting
2. Bootstrapped Aggregation (Bagging)
3. AdaBoost
4. Stacked Generalization (blending)
5. Gradient Boosting Machines (GBM)
6. Gradient Boosted Regression Trees (GBRT)
7. Random Forest

Citations

References

1. Machine Learning Algorithms



Ensemble Algorithms

CHAPTER 3

Bias and Variance

3.1 Bias

It is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

3.2 Variance

It is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

3.3 Differences

1. Bias is the algorithm's tendency to consistently learn the wrong thing by not taking into account all the information in the data (underfitting). Variance is the algorithm's tendency to learn random things irrespective of the real signal by fitting highly flexible models that follow the error/noise in the data too closely (overfitting).
2. Bias is also used to denote by how much the average accuracy of the algorithm changes as input/training data changes. Similarly, Variance is used to denote how sensitive the algorithm is to the chosen input data. Bias is prejudice in favor of or against one thing, person, or group compared with another, usually in a way considered to be unfair. Variance is the state or fact of disagreeing or quarreling.

3.4 Mathematical Representation

Let the variable we are trying to predict as y and other covariates as x. We assume there is a relationship between the two such that

$$y = f(x) + e$$

The expected squared error at a point x is:

$$Error(x) = E[(y - \hat{f}(x))^2]$$

The $Err(x)$ can be further decomposed as :

$$\begin{aligned} Error(x) &= \text{Bias}^2 + \text{Variance} + \text{IrreducibleError} \\ &= (\text{E}[\hat{f}(x)] - f(x))^2 + \text{E}[(\hat{f}(x) - \text{E}[\hat{f}(x)])^2] + \sigma_e^2 \end{aligned} \tag{3.1}$$

where

- e is the error term and it's normally distributed with a mean of 0.
- f = Target function
- $\hat{f}$ = estimation of Target function

3.4.1 Irreducible error

$$IrreducibleError = \sigma_e^2 \tag{3.2}$$

The above error can't be reduced by creating good models. It is a measure of the amount of noise in our data. Here it is important to understand that no matter how good we make our model, our data will have certain amount of noise or irreducible error that can not be removed.

3.4.2 Bias error

$$BiasError = E[\hat{f}(x)]f(x) \tag{3.3}$$

The above equation is little confusing because we can learn only one estimate for the target function $\hat{f}$ using the data we sampled, but the above equation takes expectation for $\hat{f}$. Assume that we sampled a data for n times and make a model for each sampled data. We can't expect same data every time due to irreducible error influence in the target function. As the data changes every time, our estimation of target function also change every time.

Note: Bias will be zero if, $E[\hat{f}(x)] = f(x)$. This is not possible if we make assumptions to learn the target function.

Most of the parametric methods make assumption(s) to learn a target function. The methods which make more assumptions to learn a target function are high biased method. Similarly, the methods which make very less assumptions to learn a target function are low biased method.

- Examples of low-bias machine learning algorithms: Decision Trees, k-Nearest Neighbors and Support Vector Machines.
- Examples of high-bias machine learning algorithms: Linear Regression, Linear Discriminant Analysis and Logistic Regression

3.4.3 Variance error

$$VarianceError = E[(\hat{f}(x)E[\hat{f}(x)])^2] \tag{3.4}$$

As mentioned before, for different data set, we will get different estimation for the target function. The variance error measure how much our target function $(\hat{f})$ would differ if a new training data was used. For example, let the target fucntion be given as $f = \beta_0 + \beta_1 X$; if we use regression method to learn the given target function and assume the same functional form to estimate the target function, then the number of possible estimated function will be limited. Even though we get different $(\hat{f})$ for different training data, our search space is limited due to functional form.

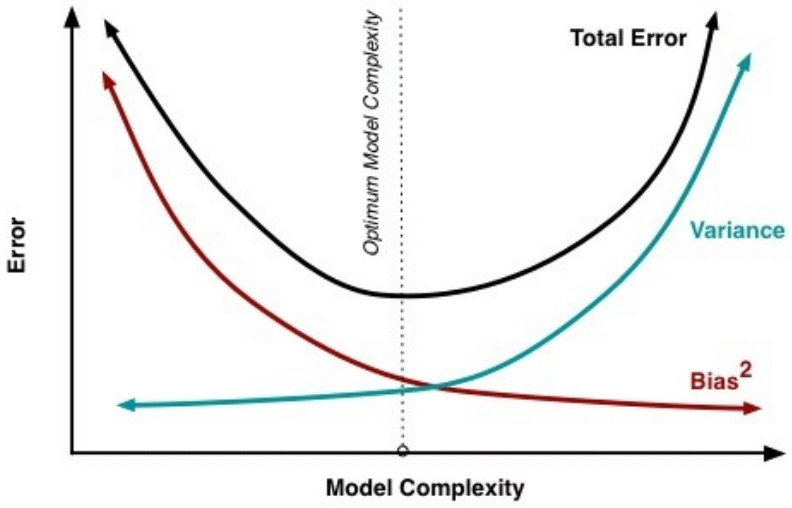
Note: If we use K-Nearest Neighbor algorithm, KNN algorithm search the estimation for target function in large dimensional space.

- If we sample different training data for the same variables and the estimated function suggests small changes from the previous $(\hat{f})$, then our model is low variance one.
- If we sample different training data for the same variables and the estimated function suggests large changes from the previous $(\hat{f})$, then our model is high variance one.

3.5 Bias-Variance Tradeoff

If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data. This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time.

At its root, dealing with bias and variance is really about dealing with over- and under-fitting. Bias is reduced and variance is increased in relation to model complexity. As more and more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls. For example, as more polynomial terms are added to a linear regression, the greater the resulting model's complexity will be. In other words, bias has a negative first-order derivative in response to model complexity while variance has a positive slope.



Understanding bias and variance is critical for understanding the behavior of prediction models, but in general what you really care about is overall error, not the specific decomposition. The sweet spot for any model is the level of

complexity at which the increase in bias is equivalent to the reduction in variance.

$$\frac{dBias}{dComplexity} = \frac{dVariance}{dComplexity} \tag{3.5}$$

If our model complexity exceeds this sweet spot, we are in effect over-fitting our model; while if our complexity falls short of the sweet spot, we are under-fitting the model. In practice, there is not an analytical way to find this location. Instead we must use an accurate measure of prediction error and explore differing levels of model complexity and then choose the complexity level that minimizes the overall error. A key to this process is the selection of an accurate error measure as often grossly inaccurate measures are used which can be deceptive.

Citations

References

1. Bias Variance

CHAPTER 4

Covariance and Correlation

Covariance and correlation describe how two variables are related.

- Variables are positively related if they move in the same direction.
- Variables are inversely related if they move in opposite directions.

Both covariance and correlation indicate whether variables are positively or inversely related. Correlation also tells you the degree to which the variables tend to move together.

4.1 Covariance

Covariance measures how two variables move with respect to each other and is an extension of the concept of variance (which tells about how a single variable varies). It can take any value from $-\infty$ to $+\infty$

- Higher this value, more dependent is the relationship. A positive number signifies positive covariance and denotes that there is a direct relationship. Effectively this means that an increase in one variable would also lead to a corresponding increase in the other variable provided other conditions remain constant.
- On the other hand, a negative number signifies negative covariance which denotes an inverse relationship between the two variables. Though covariance is perfect for defining the type of relationship, it is bad for interpreting its magnitude.

$$\mathrm{COV}_{\mathrm{x,y}} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1} \tag{4.1}$$

where,

- x = the independent variable
- y = the dependent variable
- n = number of data points in the sample
- $\bar{x}$ = the mean of the independent variable x

- $\bar{y}$ = the mean of the dependent variable y

4.2 Correlation

Correlation is another way to determine how two variables are related. In addition to telling you whether variables are positively or inversely related, correlation also tells you the degree to which the variables tend to move together.

Correlation standardizes the measure of interdependence between two variables and, consequently, tells you how closely the two variables move. The correlation measurement, called a correlation coefficient, will always take on a value between 1 and ˘1

- If the correlation coefficient is 1, the variables have a perfect positive correlation. This means that if one variable moves a given amount, the second moves proportionally in the same direction. A positive correlation coefficient less than one indicates a less than perfect positive correlation, with the strength of the correlation growing as the number approaches one.
- If correlation coefficient is 0, no relationship exists between the variables. If one variable moves, you can make no predictions about the movement of the other variable; they are uncorrelated.
- If correlation coefficient is ˘1, the variables are perfectly negatively correlated (or inversely correlated) and move in opposition to each other. If one variable increases, the other variable decreases proportionally. A negative correlation coefficient greater than ˘1 indicates a less than perfect negative correlation, with the strength of the correlation growing as the number approaches ˘1.

$$\mathrm{COR}_{\mathrm{x,y}} = \frac{\mathrm{COV}_{\mathrm{x,y}}}{\sigma_x \sigma_y} \tag{4.2}$$

where,

- σ_x = sample standard deviation of the random variable x
- σ_y = sample standard deviation of the random variable y

4.3 Difference

1. **Meaning**
   - Covariance is an indicator of the extent to which two random variables are dependent on each other. A higher number denotes higher dependency.
   - Correlation is an indicator of how strongly these two variables are related provided other conditions are constant. A maximum value is $+1$ denoting perfect dependent relationship.
2. **Relationship**
   - Correlation can be deduced from covariance.
   - Correlation provides a measure of covariance on a standard scale. It is deduced by dividing the calculated covariance with standard deviation.
3. **Value**
   - The value of covariance lies in the range of $-\infty$ and $+\infty$.
   - Correlation is limited to values between the range -1 and $+1$.
4. **Scalability**

- Correlation is not affected by a change in scales or multiplication by a constant.
- Covariance affects Correlation

5. **Units**
    - Covariance has a definite unit as it is deduced by the multiplication of two numbers and their units.
    - Correlation is a unitless absolute number between -1 and $+1$ including decimal values.

Citations

References

1. Covariance Correlation

CHAPTER 5

Model Metrics

Evaluating your machine learning algorithm is an essential. Your model may give you satisfying results when evaluated using a metric say *accuracy_score* but may give poor results when evaluated against other metrics such as *logarithmic_loss* or any other such metric. Most of the times we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model. Different performance metrics are used to evaluate different Machine Learning Algorithms. We can use classification performance metrics such as *Log-Loss*, *Accuracy*, *AUC(Area under Curve)* etc. Another example of metric for evaluation of machine learning algorithms is *precision*, *recall*, which can be used for sorting algorithms primarily used by search engines.

The metrics that you choose to evaluate your machine learning model is very important. Choice of metrics influences how the performance of machine learning algorithms is measured and compared.

5.1 Mean Absolute Error

Mean Absolute Error(MAE) is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output. However, they don't gives us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data.

If $\hat{y}_i$ is the predicted value of the i^{th} sample, and y_i is the corresponding true value, then the mean absolute error (MAE) estimated over n_{samples} is defined as:

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| \tag{5.1}$$

5.2 Mean Squared Error

Mean Squared Error(MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the square of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient. As, we take square of the error, the effect of larger errors become more pronounced then smaller error, hence the model can now focus more on the larger errors.

If $\hat{y}_i$ is the predicted value of the i^{th} sample, and y_i is the corresponding true value, then the mean absolute error (MAE) estimated over n_{samples} is defined as:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2 \tag{5.2}$$

5.3 Log Loss

Log loss, also called *logistic regression* loss or *cross-entropy loss*, is defined on probability estimates. It is commonly used in *(multinomial)* logistic regression and neural networks, as well as in some variants of expectation-maximization, and can be used to evaluate the probability outputs of a model instead of its discrete predictions.

5.3.1 Binary Classification

For binary classification with a true label $y \in \{0, 1\}$ and a probability estimate $p = \Pr(y = 1)$, the log loss per sample is the negative log-likelihood of the classifier given the true label:

$$L_{\log}(y, p) = -\log \Pr(y|p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{5.3}$$

5.3.2 Multiclass Classification

Let the true labels for a set of samples be encoded as a 1-of-K binary indicator matrix Y, i.e., $y_{i,k} = 1$ if sample i has label k taken from a set of K labels. Let P be a matrix of probability estimates, with $p_{i,k} = \Pr(t_{i,k} = 1)$. Then the log loss of the whole set is:

$$L_{\log}(Y, P) = -\log \Pr(Y|P) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} y_{i,k} \log p_{i,k} \tag{5.4}$$

where,

- $y_{i,k}$, indicates whether sample i belongs to class k or not
- $p_{i,k}$, indicates the probability of sample i belonging to class j

---

Note: To see how this generalizes the binary log loss given above, note that in the binary case, $p_{i,0} = 1 - p_{i,1}$ and $y_{i,0} = 1 - y_{i,1}$, so expanding the inner sum over $y_{i,k} \in \{0, 1\}$ gives the binary log loss. Log Loss has no upper bound and it exists on the range $[0, \infty)$. Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy. In general, minimizing Log Loss gives greater accuracy for the classifier.

In simpler terms, Log Loss works by penalizing the false classifications. It works well for multi-class classification.

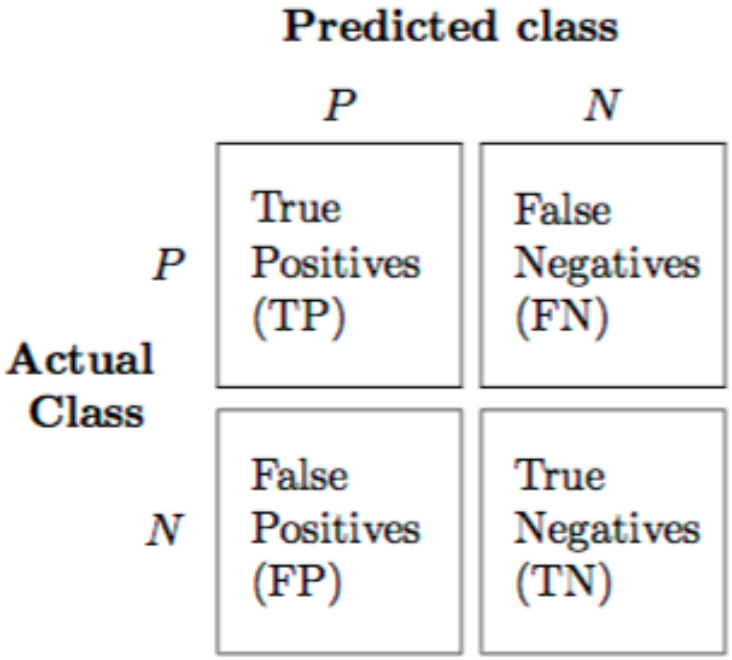
5.4 Confusion Matrix

The *Confusion matrix* is one of the most intuitive and easiest metric used for finding the correctness and accuracy of the model. It is used for Classification problem where the output can be of two or more types of classes. It in itself is not a performance measure as such, but almost all of the performance metrics are based on Confusion Matrix.

Lets say we have a classification problem where we are predicting if a person has cancer or not.

- **P :** a person tests *positive* for cancer
- **N :** a person tests *negative* for cancer

The confusion matrix, is a table with two dimensions ("Actual" and "Predicted"), and sets of "classes" in both dimensions. Our Actual classifications are rows and Predicted ones are Columns.



5.4.1 Key Terms

1. **True Positives (TP):**
   - True positives are the cases when the actual class of the data point was *Positive* and the predicted is also *Positive*. Ex: The case where a person is actually having cancer (*Positive*) and the model classifying his case as cancer (*Positive*) comes under True positive.
2. **True Negatives (TN):**
   - True negatives are the cases when the actual class of the data point was *Negative* and the predicted is also *Negative*. Ex: The case where a person NOT having cancer and the model classifying his case as Not cancer comes under True Negatives.
3. **False Positives (FP):**
   - False positives are the cases when the actual class of the data point was *Negative* and the predicted is *Positive*. False is because the model has predicted incorrectly and positive because the class predicted was a positive one. (*Positive*). Ex: A person NOT having cancer and the model classifying his case as cancer comes under False Positives.
4. **False Negatives (FN):**
   - False negatives are the cases when the actual class of the data point was *Positive* (True) and the predicted is *Negative*. False is because the model has predicted incorrectly and negative because the class predicted was a negative one. (*Negative*). Ex: A person having cancer and the model classifying his case as No-cancer comes under False Negatives.

5.5 Classification Accuracy

Classification Accuracy is what we usually mean, when we use the term accuracy. If $\hat{y}_i$ is the predicted value of the i^{th} sample and y_i is the corresponding true value, then the fraction of correct predictions over n_{samples} is defined as:

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i) \tag{5.5}$$

In short, it is the ratio of number of correct predictions to the total number of input samples:

$$\text{accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions made}} = \frac{\text{TP + TN}}{\text{TP+TN+FP+FN}} \tag{5.6}$$

1. Accuracy is a good measure when the target variable classes in the data are nearly balanced. Ex: 60% classes in our fruits images data are apple and 40% are oranges. A model which predicts whether a new image is Apple or an Orange, 97% of times correctly is a very good measure in this example.
2. Accuracy should NEVER be used as a measure when the target variable classes in the data are a majority of one class. Ex: In a cancer detection example with 100 people, only 5 people has cancer. Let's say our model is very bad and predicts every case as No Cancer. In doing so, it has classified those 95 non-cancer patients correctly and 5 cancerous patients as Non-cancerous. Now even though the model is terrible at predicting cancer, The accuracy of such a bad model is also 95%.

5.6 Precision

Using our cancer detection example, *Precision* is a measure that tells us what proportion of patients that we diagnosed as having cancer, actually had cancer. The predicted positives (People predicted as cancerous are TP and FP) and the people actually having a cancer are TP.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \tag{5.7}$$

Ex: In our cancer example with 100 people, only 5 people have cancer. Let's say our model is very bad and predicts every case as Cancer. Since we are predicting everyone as having cancer, our denominator(True positives and False Positives) is 100 and the numerator, person having cancer and the model predicting his case as cancer is 5. So in this example, we can say that Precision of such model is 5%.

5.7 Recall or Sensitivity

Recall is a measure that tells us what proportion of patients that actually had cancer was diagnosed by the algorithm as having cancer. The actual positives (People having cancer are TP and FN) and the people diagnosed by the model having a cancer are TP. (Note: FN is included because the Person actually had a cancer even though the model predicted otherwise).

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{5.8}$$

Ex: In our cancer example with 100 people, 5 people actually have cancer. Let's say that the model predicts every case as cancer. So our denominator(True positives and False Negatives) is 5 and the numerator, person having cancer and the model predicting his case as cancer is also 5(Since we predicted 5 cancer cases correctly). So in this example, we can say that the Recall of such model is 100%. And Precision of such a model(As we saw above) is 5%.

1. **Precision** is about being precise. So even if we managed to capture only one cancer case, and we captured it correctly, then we are 100% precise.
2. **Recall** is not so much about capturing cases correctly but more about capturing all cases that have "cancer" with the answer as "cancer". So if we simply always say every case as "cancer", we have 100% recall.

5.8 F1 Score

F1 Score, also known as the *Sørensen–Dice coefficient* or *Dice similarity coefficient(DSC)*, is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model.

$$F_1 = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5.9}$$

Two other commonly used F measures are the F_2 measure, which weighs recall higher than precision (by placing more emphasis on false negatives), and the $F_{0.5}$ measure, which weighs recall lower than precision (by attenuating the influence of false negatives).

5.9 Receiver operating characteristic (ROC)

Receiver operating characteristic (ROC), or simply *ROC curve*, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate.

5.9.1 True Positive Rate (Sensitivity)

True Positive Rate corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points.

$$\text{TruePositiveRate} = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \tag{5.10}$$

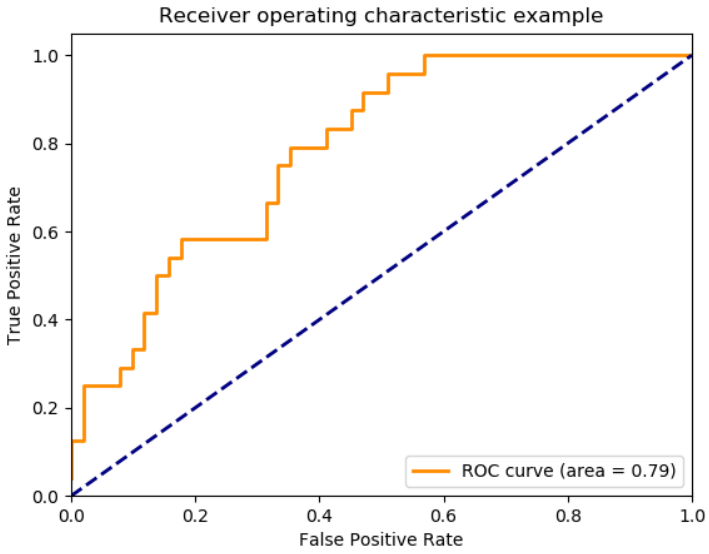
5.9.2 False Positive Rate (Specificity)

False Positive Rate corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

$$\text{FalsePositiveRate} = \frac{\text{False Positive}}{\text{False Positive + True Negative}} \tag{5.11}$$

This function requires the true binary value and the target scores, which can either be probability estimates of the positive class, confidence values, or binary decisions. False Positive Rate and True Positive Rate both have values in the range [0, 1]. FPR and TPR both are computed at threshold values such as (0.00, 0.02, 0.04, ..., 1.00) and a graph is drawn.

Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. To compute the points in an ROC curve, we could evaluate a logistic regression model many times with different classification thresholds.



5.10 AUC: Area Under the ROC Curve

Area under the ROC Curve, is the measure of an entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1). AUC provides an aggregate measure of performance across all possible classification thresholds. AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. Ex: given the following examples, which are arranged from left to right in ascending order of logistic regression predictions; AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red).

5.10.1 Properties

1. AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
2. AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.
3. **Scale invariance is not always desirable**. For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
4. **Classification-threshold invariance is not always desirable**. In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. AUC isn't a useful metric for this type of optimization.

		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	F_1 score = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$

Citations

References

1. Metrics Evaluation
2. Metrics Performance

CHAPTER 6

Underfitting and Overfitting

In machine learning we describe the learning of the target function from training data as inductive learning. Induction refers to learning general concepts from specific examples which is exactly the problem that supervised machine learning problems aim to solve. This is different from deduction that is the other way around and seeks to learn specific concepts from general rules.

Generalization refers to how well the concepts learned by a machine learning model apply to specific examples not seen by the model when it was learning. The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen.

There is a terminology used in machine learning when we talk about how well a machine learning model learns and generalizes to new data, namely overfitting and underfitting. Overfitting and underfitting are the two biggest causes for poor performance of machine learning algorithms.

6.1 Overfitting

Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

Overfitting occurs when a statistical model or machine learning algorithm captures the noise of the data. Intuitively, overfitting occurs when the model or the algorithm fits the data too well. Specifically, overfitting occurs if the model or algorithm shows low bias but high variance. Overfitting is often a result of an excessively complicated model, and it can be prevented by fitting multiple models and using validation or cross-validation to compare their predictive accuracies on test data.

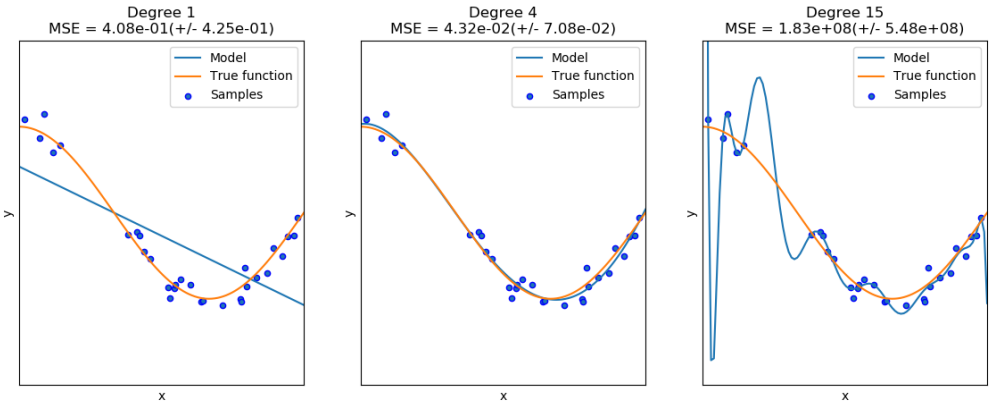
6.2 Underfitting

Underfitting refers to a model that can neither model the training data nor generalize to new data. An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data. Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.

Underfitting occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data. Intuitively, underfitting occurs when the model or the algorithm does not fit the data well enough. Specifically, underfitting occurs if the model or algorithm shows low variance but high bias. Underfitting is often a result of an excessively simple model.

6.3 Example

This example demonstrates the problems of underfitting and overfitting and how we can use linear regression with polynomial features to approximate nonlinear functions. The plot shows the function that we want to approximate, which is a part of the cosine function. In addition, the samples from the real function and the approximations of different models are displayed. The models have polynomial features of different degrees. We can see that a linear function (polynomial with degree 1) is not sufficient to fit the training samples. This is called underfitting. A polynomial of degree 4 approximates the true function almost perfectly. However, for higher degrees the model will overfit the training data, i.e. it learns the noise of the training data. We evaluate quantitatively overfitting / underfitting by using cross-validation. We calculate the mean squared error (MSE) on the validation set, the higher, the less likely the model generalizes correctly from the training data.



Citations

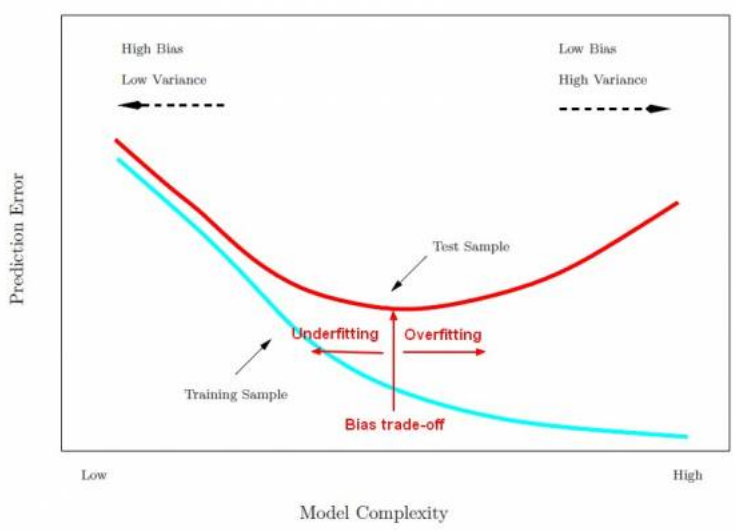
References

1. Underfitting vs Overfitting

CHAPTER 7

Model Performance

When training ML Models, it is important to avoid *overfitting* the training data. Overfitting occurs when the ML model learns the noise in the training data and thus does not generalize well to data it has not been trained on. One *hyperparameter* that affects whether the ML model will overfit or not is the number of epochs or complete passes through the training split. If we use too many epochs, then the ML model is likely to overfit. On the other hand, if we use too few epochs, the ML model might not have the chance to learn fully from the training data.



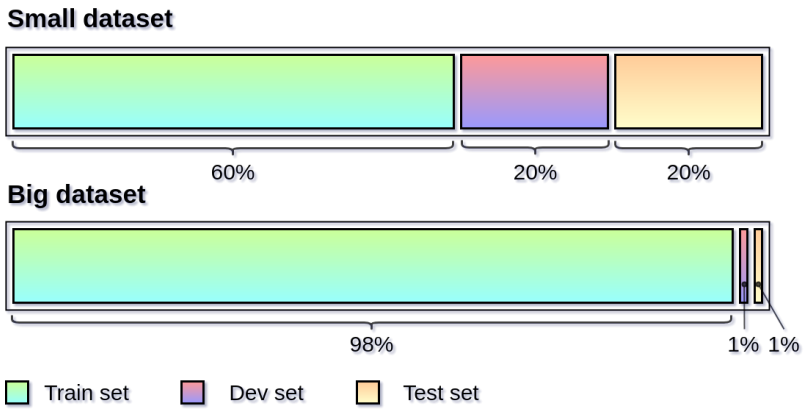
Important terms

- **Training data** is the data given to the model, from which the model build your model.
- **Training accuracy** tells about how much your model learns to map the input and output.
- **Validation data** is the data with which the training process is validated.
- **Validation accuracy** tells about the generalizing power of the model.

- **Validation accuracy** decreasing means lower generalization over the training data. Validation accuracy is evaluated while Training.
- **Test data** assess the performance of a trained model.
- **Test accuracy** gives the final generalization power. Test accuracy is evaluated after training.
- If *training and validation* are both low, you are probably under fitting and you can probably increase the capacity of your model and train more or longer (increase the number of epochs).

7.1 Data Splitting

In practice, detecting that our model is overfitting is difficult. It's not uncommon that our trained model is already in production and then we start to realize that something is wrong. In fact, it is only by confronting new data that you can make sure that everything is working properly. However, during the training we should try to reproduce the real conditions as much as possible. For this reason, it is good practice to *divide* our dataset into three parts - **training set**, **dev set (also known as cross-validation or hold-out)** and **test set**. Our model learns by seeing only the first of these parts. Hold-out is used to track our progress and draw conclusions to optimize the model. While, we use a test set at the end of the training process to evaluate the performance of our model.



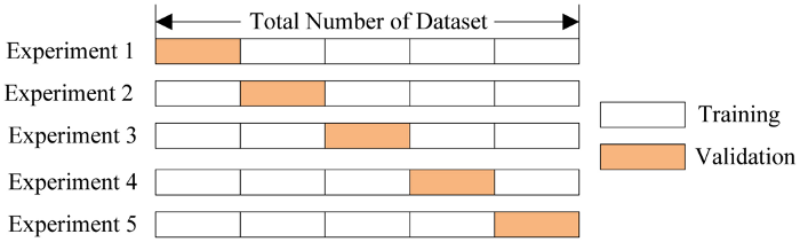
It is very important to make sure that your cross-validation and test set come from the same distribution as well as that they accurately reflect data that we expect to receive in the future. Only then we can be sure that the decisions we make during the learning process bring us closer to a better solution. Our dev and test sets should be simply large enough to give us high confidence in the performance of our model.

7.2 Validation

We need to create a model with the best settings (the degree), but we don't want to have to keep going through training and testing. There are no consequences in our example from poor test performance, but in a real application where we might be performing a critical task such as diagnosing cancer, there would be serious downsides to deploying a faulty model. We need some sort of pre-test to use for model optimization and evaluate. This pre-test is known as a validation set. A basic approach would be to use a validation set in addition to the training and testing set. This presents a few problems though: we could just end up overfitting to the validation set and we would have less training data. A smarter implementation of the validation concept is k-fold cross-validation.

The idea is straightforward: rather than using a separate validation set, we split the training set into a number of subsets, called folds. Let's use five folds as an example. We perform a series of train and evaluate cycles where each time we train on 4 of the folds and test on the 5th, called the hold-out set. We repeat this cycle 5 times, each time using a different fold for evaluation. At the end, we average the scores for each of the folds to determine the

overall performance of a given model. This allows us to optimize the model before deployment without having to use additional data.



7.3 Cost function

Whenever a model is trained on a training data and is used to predict values on a testing set, there exists a difference between the true and predicted values. The closer the predicted values to their corresponding real values, the better the model. That means, a **cost function** is used to measure how close the predicted values are to their corresponding real values. The function can be minimized or maximized, given the situation/problem.

For example, in case of ordinary least squares (OLS), the cost function(to be minimized) would be:

$$\mathrm{J}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 \tag{7.1}$$

where,

- J denotes the cost function,
- m is the number of observations in the dataset,
- $h(x)$ is the predicted value of the response
- y is the true value of the response

7.4 High Bias and High Variance

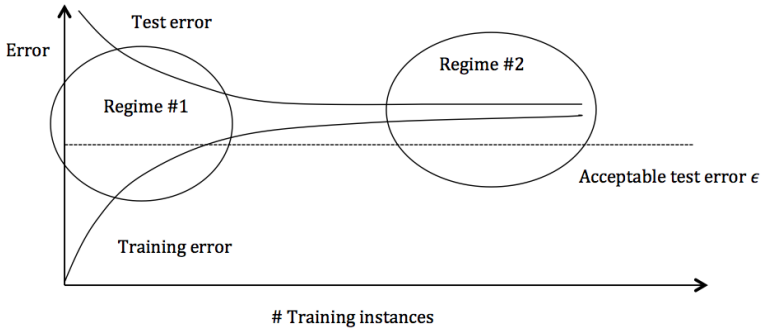
If a model is under-performing (e.g. if the test or training error is too high), there are several ways to improve performance. To find out which of these many techniques is the right one for the situation, the first step is to determine the root of the problem.

The graph above plots the training error and the test error and can be divided into two overarching regimes. In the first regime (on the left side of the graph), training error is below the desired error threshold (denoted by), but test error is significantly higher. In the second regime (on the right side of the graph), test error is remarkably close to training error, but both are above the desired tolerance of .

7.4.1 Regime 1 (High Variance)

In the first regime, the cause of the poor performance is high variance.

- **Symptoms**
  1. Training error is much lower than test error



2. Training error is lower than
3. Test error is above

- **Remedies**
  1. Add more training data
  2. Reduce model complexity – complex models are prone to high variance
  3. Bagging (will be covered later in the course)

7.4.2 Regime 2 (High Bias)

Unlike the first regime, the second regime indicates high bias: the model being used is not robust enough to produce an accurate prediction.

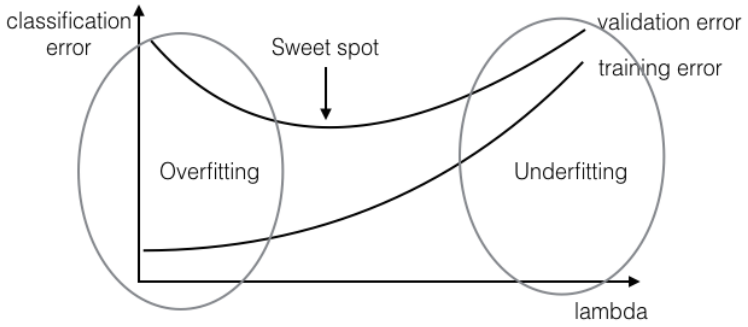
- **Symptoms**
  1. Training error is higher than
- **Remedies**
  1. Use more complex model (e.g. kernelize, use non-linear models)
  2. Add features
  3. Boosting

7.5 Regularizations

One of the first methods we should try when we need to reduce overfitting is *regularization*. It involves adding an extra element to the loss function, which punishes our model for being too complex or, in simple words, for using too high values in the weight matrix. This way we try to limit its flexibility, but also encourage it to build solutions based on multiple features. Two popular versions of this method are:

1. **L1 regularization (Lasso Regression)** : *(Least Absolute Shrinkage and Selection Operator)* adds *absolute value of magnitude* of coefficient as penalty term to the loss function.

$$\mathrm{J}(\theta)^{L1} = \mathrm{J}(\theta)^{OLS} + \lambda \sum_{j=1}^{n} |\theta_j| \tag{7.2}$$



1. **L2 Regularization (Ridge regression)** : adds *squared magnitude* of coefficient as penalty term to the loss function.

$$\mathrm{J}(\theta)^{L2} = \mathrm{J}(\theta)^{OLS} + \lambda \sum_{j=1}^{n} \theta_j^2 \tag{7.3}$$

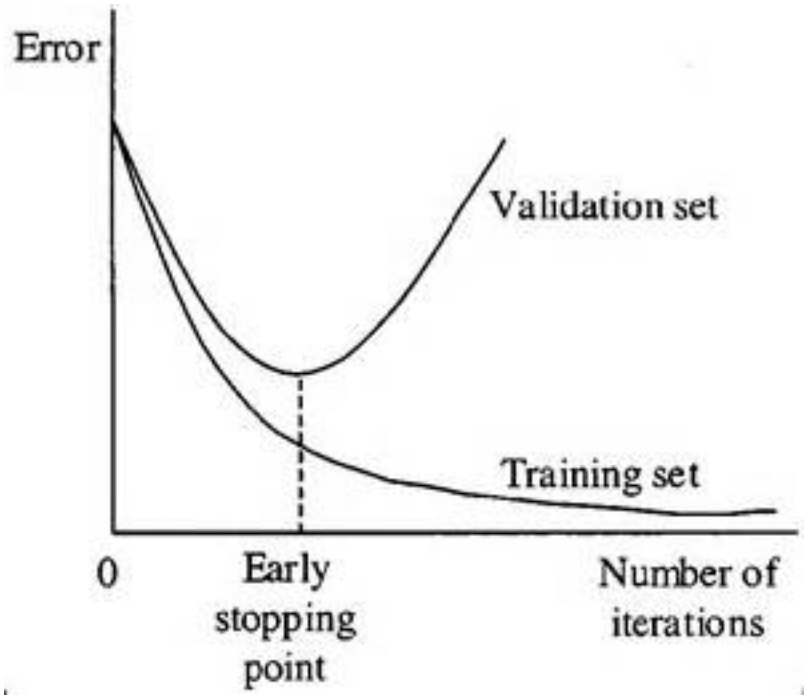
In addition to the cost function we had in case of OLS, there is an additional term added, which is the regularization term. θ(norm of the coefficients), the addition is of λ(the regularization parameter) and θ^2(norm of coefficient squared). The addition of regularization term penalizes big coefficients and tries to minimize them to zero, although not making them exactly to zero. This means that if the $'s$ take on large values, the optimization function is penalized. We would prefer to take smaller 's, or 's that are close to zero to drive the penalty term small.

7.5.1 Key points

1. Lasso shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features.
2. Built-in feature selection is frequently mentioned as a useful property of the L1-norm, which the L2-norm does not. This is actually a result of the L1-norm, which tends to produces sparse coefficients.
3. Computational efficiency, L1-norm does not have an analytical solution, but L2-norm does. This allows the L2-norm solutions to be calculated computationally efficiently. However, L1-norm solutions does have the sparsity properties which allows it to be used along with sparse algorithms, which makes the calculation more computationally efficient.
4. When there are many predictors (with some col-linearity among them) in the dataset and not all of them have the same predicting power, L2 regression can be used to estimate the predictor importance and penalize predictors that are not important. One issue with co-linearity is that the variance of the parameter estimate is huge. In cases where the number of features are greater than the number of observations, the matrix used in the OLS may not be invertible but Ridge Regression enables this matrix to be inverted. It seeks to reduce the MSE by adding some bias and, at the same time, reducing the variance. Remember high variance correlates to a over-fitting model.
5. One of the things that Ridge can't be used is variable selection since it retains all the predictors. Lasso on the other hand overcomes this problem by forcing some of the predictors to zero.
6. As the λ is increased, variance is reduced and bias is added in the model, so getting the right value of the lambda is essential. Cross-validation is generally used to estimate the value of lambda.

7.6 Early Stopping

When you're training a learning algorithm iteratively, you can measure how well each iteration of the model performs. Up until a certain number of iterations, new iterations improve the model. After that point, however, the model's ability to generalize can weaken as it begins to overfit the training data. Early stopping refers stopping the training process before the learner passes that point.



Today, this technique is mostly used in deep learning while other techniques (e.g. regularization) are preferred for classical machine learning.

7.7 Hyperparameter Optimization

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data.[1] The objective function takes a tuple of hyperparameters and returns the associated loss. Cross-validation is often used to estimate this generalization performance.

7.7.1 Grid search

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.

7.7.2 Random search

Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm

7.7.3 Bayesian optimization

Bayesian optimization is a global optimization method for noisy black-box functions. Applied to hyperparameter optimization, Bayesian optimization builds a probabilistic model of the function mapping from hyperparameter values to the objective evaluated on a validation set. By iteratively evaluating a promising hyperparameter configuration based on the current model, and then updating it, Bayesian optimization, aims to gather observations revealing as much information as possible about this function and, in particular, the location of the optimum. It tries to balance exploration (hyperparameters for which the outcome is most uncertain) and exploitation (hyperparameters expected close to the optimum).

7.7.4 Gradient-based optimization

For specific learning algorithms, it is possible to compute the gradient with respect to hyperparameters and then optimize the hyperparameters using gradient descent. The first usage of these techniques was focused on neural networks. Since then, these methods have been extended to other models such as support vector machines or logistic regression.

Citations

References

1. Model Performance
2. Model Evaluation

CHAPTER 8

Gradient descent

Gradient Descent is a method used while training a machine learning model. It is an optimization algorithm, based on a convex function, that tweaks it's parameters iteratively to minimize a given function to its local minimum. It is simply used to find the values of a functions parameters (coefficients) that minimize a cost function as far as possible. You start by defining the initial parameters values and from there on Gradient Descent iteratively adjusts the values, using calculus, so that they minimize the given cost-function

8.1 Gradient

A gradient measures how much the output of a function changes if you change the inputs a little bit. It simply measures the change in all weights with regard to the change in error. You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning. Said it more mathematically, a gradient is a partial derivative with respect to its inputs.

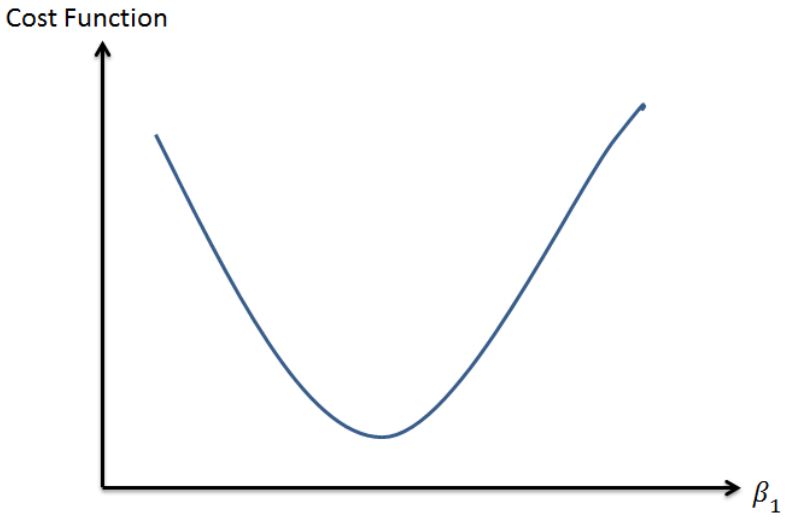
8.2 Cost Function

It is a way to determine how well the machine learning model has performed given the different values of each parameters. The linear regression model, the parameters will be the two coefficients, β and m :

$$y = \beta + mx_1$$

The cost function will be the sum of least square methods. Since the cost function is a function of the parameters β and m, we can plot out the cost function with each value of the coefficients. (i.e. Given the value of each coefficient, we can refer to the cost function to know how well the machine learning model has performed). The cost function looks like:

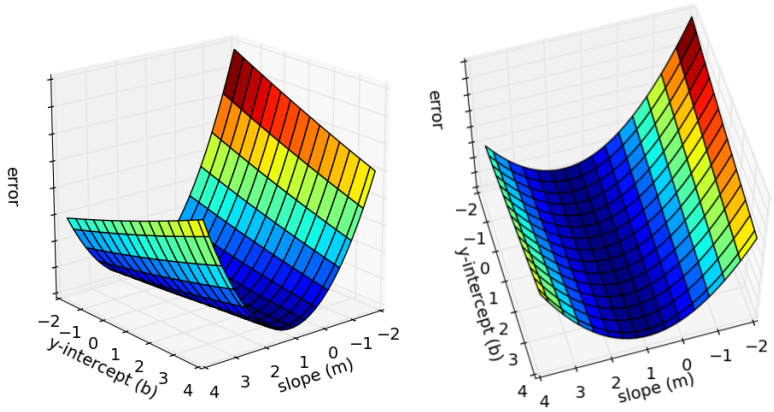
1. during the training phase, we are focused on selecting the 'best' value for the parameters (i.e. the coefficients), x's will remain the same throughout the training phase
2. for the case of linear regression, we are finding the value of the coefficients that will reduce the cost to the minimum a.k.a the lowest point in the mountainous region.



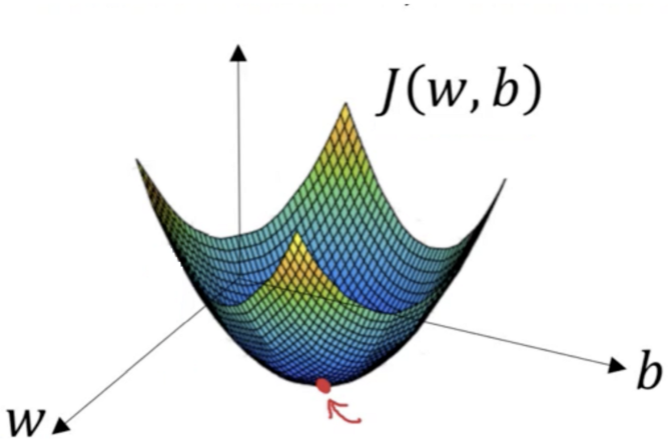
8.3 Method

The Cost function will take in a (m, b) pair and return an error value based on how well the line fits our data. To compute this error for a given line, we'll iterate through each (x, y) point in our data set and sum the square distances between each point's y value and the candidate line's y value *(computed at mx + b)*. It's conventional to square this distance to ensure that it is positive and to make our error function differentiable.

- Lines that fit our data better (where better is defined by our error function) will result in lower error values. If we minimize this function, we will get the best line for our data. Since our error function consists of two parameters (m and b) we can visualize it as a two-dimensional surface.



- Each point in this two-dimensional space represents a line. The height of the function at each point is the error value for that line. Some lines yield smaller error values than others (i.e., fit our data better). When we run gradient descent search, we will start from some location on this surface and move downhill to find the line with the lowest error.
- The horizontal axes represent the parameters (w and β) and the cost function $J(w, \beta)$ is represented on the vertical axes. You can also see in the image that gradient descent is a convex function.
- we want to find the values of w and β that correspond to the minimum of the cost function (marked with the red arrow). To start with finding the right values we initialize the values of w and β with some random numbers and Gradient Descent then starts at that point.

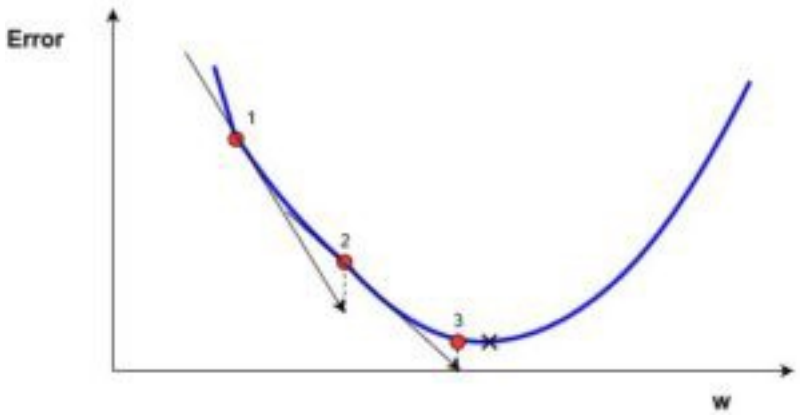


- Then it takes one step after another in the steepest downside direction till it reaches the point where the cost function is as small as possible.

8.4 Algorithm

Moving forward to find the lowest error (deepest valley) in the cost function (with respect to one weight) we need to tweak the parameters of the model. Using calculus, we know that the slope of a function is the derivative of the function with respect to a value. *This slope always points to the nearest valley.*

$$Error_{(m,\beta)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + \beta))^2 \tag{8.1}$$



We can see the graph of the cost function(named $Error$ with symbol J) against just one weight. Now if we calculate the slope(let's call this $(\frac{dJ}{dw})$ of the cost function with respect to this one weight, we get the direction we need to move towards, in order to reach the local minima(nearest deepest valley).

Note: The gradient (or derivative) tells us the incline or slope of the cost function. Hence, to minimize the cost function, we move in the direction opposite to the gradient.

8.4.1 Steps

1. Initialize the weights w randomly.
2. Calculate the gradients G of cost function w.r.t parameters. This is done using partial differentiation: $G = J(w)/w$. The value of the gradient G depends on the inputs, the current values of the model parameters, and the cost function.
3. Update the weights by an amount proportional to G, i.e. $w = w - G$
4. Repeat until the cost $J(w)$ stops reducing, or some other pre-defined termination criteria is met.

Listing 1: gradient descent / batch gradient descent

```
def calculate_cost(theta, x, y):
    """
    calculates the cost for the given X and y
    """
    m = len(y)
    predictions = X.dot(theta)
    cost = np.sum(np.square(predictions-y))/(2*m)
    return cost

    def gradient_descent(X, y, theta, learning_rate=0.01, iterations=1000):
    """
    returns the final theta vector and the array of the cost history
    """
    m = len(y)
    cost_history = np.zeros(iterations)
    theta_history = np.zeros((iterations,2))
    for it in range(iterations):
        prediction = np.dot(X, theta)
        theta -= (1/m)*learning_rate*(X.T.dot((prediction - y)))
        theta_history[it,:] = theta.T
        cost_history[it] = calculate_cost(theta, X, y)
    return theta, cost_history, theta_history
```

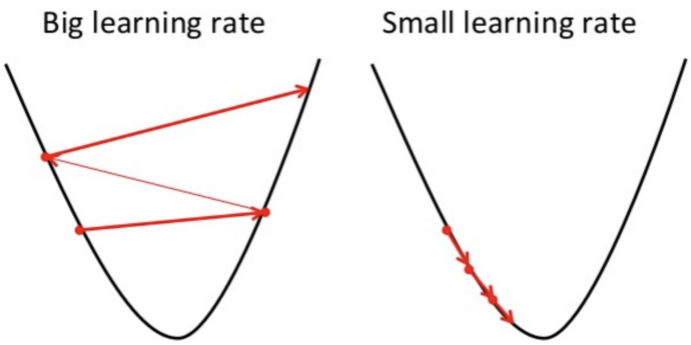
Important: In step 3, η is the learning rate which determines the size of the steps we take to reach a minimum. We need to be very careful about this parameter. High values of η may overshoot the minimum, and very low values will reach the minimum very slowly.

8.5 Learning Rate

How big the steps are that Gradient Descent takes into the direction of the local minimum are determined by the so-called **learning rate**. It determines how fast or slow we will move towards the optimal weights. In order for Gradient Descent to reach the local minimum, we have to set the learning rate to an appropriate value, which is neither too low nor too high.

This is because if the steps it takes are too big, it maybe will not reach the local minimum because it just bounces back and forth between the convex function of gradient descent like you can see on the left side of the image below. If you set the learning rate to a very small value, gradient descent will eventually reach the local minimum but it will maybe take too much time like you can see on the right side of the image.

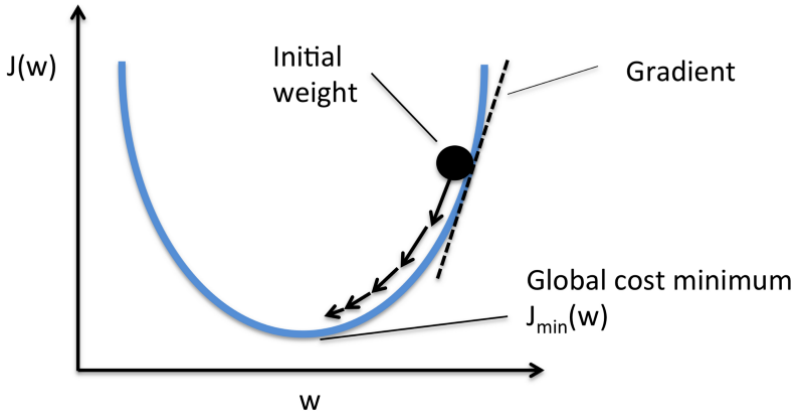
Note: When you're starting out with gradient descent on a given problem, just simply try 0.001, 0.003, 0.01, 0.03,



0.1, 0.3, 1 etc. as it's learning rates and look which one performs the best.

8.6 Convergence

Once the agent, after many steps, realize the cost does not improve by a lot and it is stuck very near a particular point (minima), technically this is known as **convergence**. The value of the parameters at that very last step is known as the 'best' set of parameters, and we have a trained model.



8.7 Types of Gradient Descent

Three popular types of Gradient Descent, that mainly differ in the amount of data they use.

8.7.1 Batch Gradient Descent

Also called vanilla gradient descent, calculates the error for each example within the training dataset, but only after all training examples have been evaluated, the model gets updated. This whole process is like a cycle and called a training epoch.

1. Advantages of it are that it's computational efficient, it produces a stable error gradient and a stable convergence.

2. Batch Gradient Descent has the disadvantage that the stable error gradient can sometimes result in a state of convergence that isn't the best the model can achieve. It also requires that the entire training dataset is in memory and available to the algorithm.

8.7.2 Stochastic gradient descent (SGD)

In vanilla gradient descent algorithms, we calculate the gradients on each observation one by one; In stochastic gradient descent we can chose the random observations randomly. It is called **stochastic** because samples are selected randomly (or shuffled) instead of as a single group (as in standard gradient descent) or in the order they appear in the training set. This means that it updates the parameters for each training example, one by one. This can make SGD faster than Batch Gradient Descent, depending on the problem.

1. One advantage is that the frequent updates allow us to have a pretty detailed rate of improvement. The frequent updates are more computationally expensive as the approach of Batch Gradient Descent.
2. The frequency of those updates can also result in noisy gradients, which may cause the error rate to jump around, instead of slowly decreasing.

Listing 2: Stochastic gradient descent

```
def stocashtic_gradient_descent(X, y, theta, learning_rate=0.01, iterations=100):
    """
    returns the final theta vector and the array of the cost history
    """
    m = len(y)
    cost_history = np.zeros(iterations)

    for it in range(iterations):
        cost = 0.0
        for i in range(m):
            rand_ind = np.random.randint(0,m)
            X_i = X[rand_ind,:].reshape(1, X.shape[1])
            y_i = y[rand_ind].reshape(1,1)
            prediction = np.dot(X_i, theta)

            theta -= (1/m)*learning_rate*(X_i.T.dot((prediction - y_i)))
            cost += calculate_cost(theta, X_i, y_i)
        cost_history[it] = cost

    return theta, cost_history, theta_history
```

8.7.3 Mini-batch Gradient Descent

Is a combination of the concepts of SGD and Batch Gradient Descent. It simply splits the training dataset into small batches and performs an update for each of these batches. Therefore it creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

1. Common mini-batch sizes range between 50 and 256, but like for any other machine learning techniques, there is no clear rule, because they can vary for different applications. It is the most common type of gradient descent within deep learning.

Listing 3: mini-batch gradient descent

```
def stocashtic_gradient_descent(X, y, theta, learning_rate=0.01, iterations=100,␣
↪batch_size=20):
    """
    returns the final theta vector and the array of the cost history
    """
    m = len(y)
    cost_history = np.zeros(iterations)
    n_batches = int(m/batch_size)

    for it in range(iterations):
    cost = 0.0
    indices = np.random.permumtation(m)
    X = X[indices]
    y = y[indices]
    for i in range(0, m, batch_size):
        X_i = X[i:i+batch_size]
        y_i = y[i:i+batch_size]
        X_i = np.c_[np.ones(len(X_i)), X_i]
        prediction = np.dot(X_i, theta)

        theta -= (1/m)*learning_rate*(X_i.T.dot((prediction - y_i)))
        cost += calculate_cost(theta, X_i, y_i)
    cost_history[it] = cost

    return theta, cost_history, theta_history
```

Citations

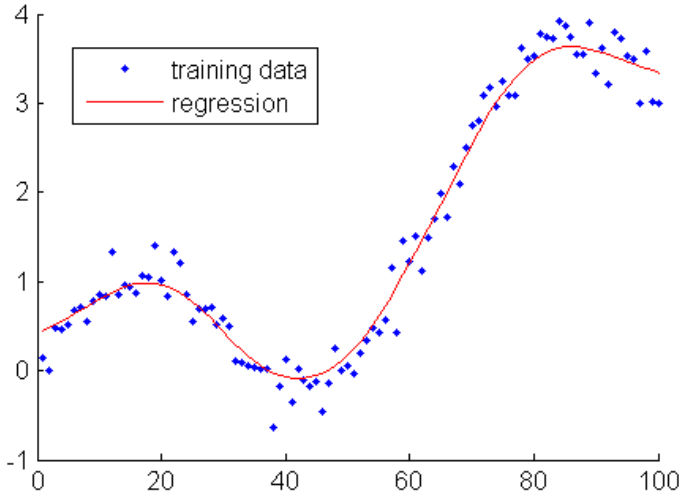
References

1. Gradient Descent
2. Gradient Descent Linear Regression

CHAPTER 9

Regression

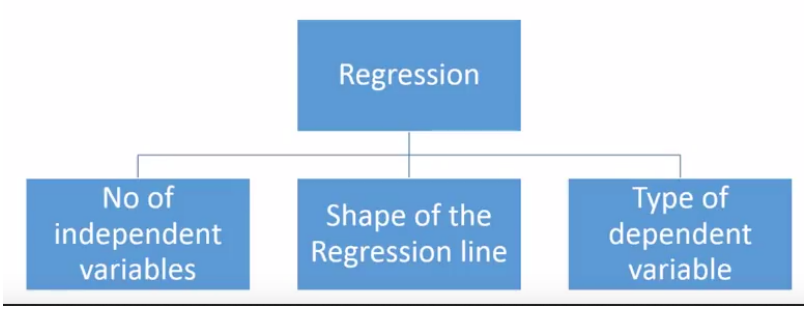
Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent variable(s) (target) and independent variable(s) (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables. Regression analysis is an important tool for modelling and analyzing data. Here, we fit a curve / line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized.



Important: One important fact to note, and one that is often brushed over or forgotten, is that statistical analysis, regression analysis included, can only ever indicate correlations between factors, not causal relationships. Regression analysis is a great technique for making predictions and understanding the influences of variables on one another, but are sometimes misused, or misunderstood, and taken to be a reliable proof of causality, this is misleading. While some of the variables included in a regression model may very well be causally related to one another, they also might not be; without empirical testing, these relationships cannot be taken as absolute.

9.1 Basic Models

There are various kinds of regression techniques available to make predictions. These techniques are mostly driven by three metrics (number of independent variables, type of dependent variables and shape of regression line).



9.1.1 Continuous variables

Continuous variables are a measurement on a continuous scale, such as weight, time, and length.

Linear regression

Linear regression, also known as ordinary least squares (OLS) and linear least squares, is the real workhorse of the regression world. Use linear regression to understand the mean change in a dependent variable given a one-unit change in each independent variable.

1. There must be linear relationship between independent and dependent variables
2. Multiple regression suffers from multicollinearity, autocorrelation, heteroskedasticity.
3. Linear Regression is very sensitive to Outliers. It can terribly affect the regression line and eventually the forecasted values.
4. Multicollinearity can increase the variance of the coefficient estimates and make the estimates very sensitive to minor changes in the model. The result is that the coefficient estimates are unstable
5. In case of multiple independent variables, we can go with forward selection, backward elimination and step wise approach for selection of most significant independent variables.

Polynomial Regression

When we want to create a model that is suitable for handling non-linearly separable data, we will need to use a polynomial regression. In this regression technique, the best fit line is not a straight line. It is rather a curve that fits into the data points.

1. Able to model non-linearly separable data; linear regression can't do this. It is much more flexible in general and can model some fairly complex relationships.
2. Full control over the modelling of feature variables (which exponent to set).
3. Requires careful design. Need some knowledge of the data in order to select the best exponents.
4. Prone to over fitting if exponents are poorly selected.

Ridge regression

Ridge Regression is a technique used when the data suffers from multicollinearity (independent variables are highly correlated). In multicollinearity, even though the least squares estimates (OLS) are unbiased, their variances are large which deviates the observed value far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.

1. It allows you to analyze data even when severe multicollinearity is present and helps prevent overfitting. This type of model reduces the large, problematic variance that multicollinearity causes by introducing a slight bias in the estimates.
2. The assumptions of this regression is same as least squared regression except normality is not to be assumed
3. It shrinks the value of coefficients but doesn't reaches zero, which suggests no feature selection feature
4. This is a regularization method and uses l2 regularization.
5. The procedure trades away much of the variance in exchange for a little bias, which produces more useful coefficient estimates when multicollinearity is present.

Lasso regression

Lasso regression (least absolute shrinkage and selection operator) performs variable selection that aims to increase prediction accuracy by identifying a simpler model. It is similar to Ridge regression but with variable selection. Similar to Ridge Regression, Lasso (Least Absolute Shrinkage and Selection Operator) also penalizes the absolute size of the regression coefficients. In addition, it is capable of reducing the variability and improving the accuracy of linear regression models. Lasso regression differs from ridge regression in a way that it uses absolute values in the penalty function, instead of squares. This leads to penalizing (or equivalently constraining the sum of the absolute values of the estimates) values which causes some of the parameter estimates to turn out exactly zero. Larger the penalty applied, further the estimates get shrunk towards absolute zero. This results to variable selection out of given n variables.

1. The assumptions of this regression is same as least squared regression except normality is not to be assumed
2. It shrinks coefficients to zero (exactly zero), which certainly helps in feature selection
3. This is a regularization method and uses l1 regularization
4. If group of predictors are highly correlated, lasso picks only one of them and shrinks the others to zero

ElasticNet Regression

ElasticNet is hybrid of Lasso and Ridge Regression techniques. It is trained with L1 and L2 prior as regularizer. Elastic-net is useful when there are multiple features which are correlated. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both. A practical advantage of trading-off between Lasso and Ridge is that, it allows Elastic-Net to inherit some of Ridge's stability under rotation.

1. It encourages group effect in case of highly correlated variables
2. There are no limitations on the number of selected variables
3. It can suffer with double shrinkage

9.1.2 Categorical variables

A categorical variable has values that you can put into a countable number of distinct groups based on a characteristic.

Binary Logistic Regression

Use binary logistic regression to understand how changes in the independent variables are associated with changes in the probability of an event occurring. This type of model requires a binary dependent variable. A binary variable has only two possible values, such as pass and fail.

Ordinal Logistic Regression

Ordinal logistic regression models the relationship between a set of predictors and an ordinal response variable. An ordinal response has at least three groups which have a natural order, such as hot, medium, and cold.

Nominal Logistic Regression

Nominal logistic regression models the relationship between a set of independent variables and a nominal dependent variable. A nominal variable has at least three groups which do not have a natural order, such as scratch, dent, and tear.

Poisson regression

Use Poisson regression to model how changes in the independent variables are associated with changes in the counts. Poisson models are similar to logistic models because they use Maximum Likelihood Estimation and transform the dependent variable using the natural log. Poisson models can be suitable for rate data, where the rate is a count of events divided by a measure of that unit's exposure (a consistent unit of observation).

9.2 Selecting Model

Within multiple types of regression models, it is important to choose the best suited technique based on type of independent and dependent variable , dimensionality in the data and other essential characteristics of the data.

1. Data exploration is an inevitable part of building predictive model. It should be you first step before selecting the right model like identify the relationship and impact of variables
2. To compare the goodness of fit for different models, we can analyse different metrics like statistical significance of parameters, R-square, Adjusted r-square, AIC, BIC and error term. Another one is the Mallow's Cp criterion. This essentially checks for possible bias in your model, by comparing the model with all possible submodels (or a careful selection of them).
3. Cross-validation is the best way to evaluate models used for prediction. Here you divide your data set into two group (train and validate). A simple mean squared difference between the observed and predicted values give you a measure for the prediction accuracy.
4. If your data set has multiple confounding variables, you should not choose automatic model selection method because you do not want to put these in a model at the same time.
5. It'll also depend on your objective. It can occur that a less powerful model is easy to implement as compared to a highly statistically significant model.
6. Regression regularization methods(Lasso, Ridge and ElasticNet) works well in case of high dimensionality and multicollinearity among the variables in the data set.

Citations

References

1. Regression Guide
2. Regression Analysis
3. Regression Types

CHAPTER 10

Simple Linear Regression

Linear regression is an approach for predicting a *quantitative response* using *feature predictor* or *input variable*. It is a basic predictive analytics technique that uses historical data to predict an output variable. It takes the following form:

$$y = \beta_0 + \beta_1 * x$$

where,

- y is the response, a function of x
- x is the feature
- $beta_0$ is the intercept, also called *bias coefficient*
- $beta_1$ is the slope, also called the *scale factor*
- $beta_0$ and β_1 are called the model coefficients

Our goal is to find statistically significant values of the parameters β_0 and β_1 that minimize the difference between y (*output*) and y_e (*estimated / predicted output*).

Note: The predicted values here are continuous in nature. So, your ultimate goal is, given a training set, to learn a function $f : X ß Y$ so that $f(x)$ is a good predictor for the corresponding value of y. Also, keep in mind that the domain of values both x and y are all real numbers

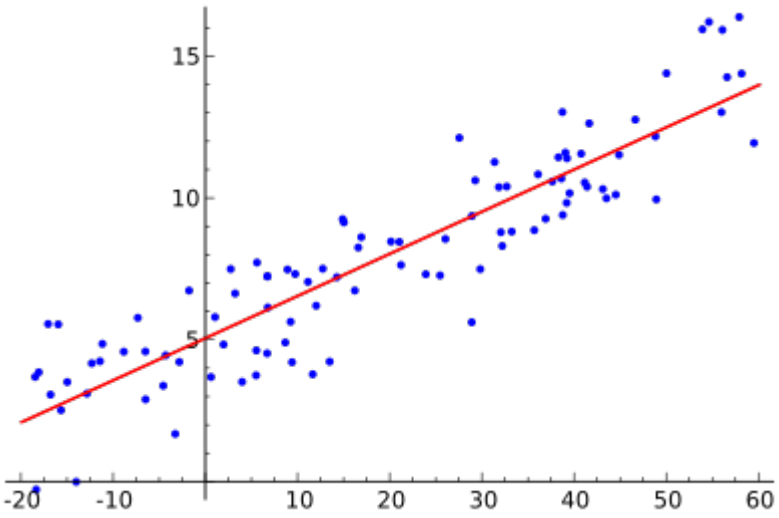
---

If we are able to determine the optimum values of these two parameters, then we will have the **Line of Best Fit** that we can use to predict the values of y, given the value of x.

It is important to note that, linear regression can often be divided into two basic forms:

1. Simple Linear Regression (SLR) which deals with just two variables (the one you saw at first)
2. Multi-linear Regression (MLR) which deals with more than two variables

So, how do we estimate β_0 and β_1? We can use a method called ordinary least squares.



10.1 Ordinary Least Sqaure

10.1.1 Method

Ordinary least squares (OLS) is a type of linear least squares method for estimating the unknown parameters in a *linear regression model*. OLS chooses the parameters of a linear function of a set of explanatory variables by the principle of least squares: minimizing the sum of the squares of the differences between the observed dependent variable (values of the variable being predicted) in the given dataset and those predicted by the linear function.

This is seen as the sum of the squared distances, parallel to the axis of the dependent variable, between each data point in the set and the corresponding point on the regression surface – the smaller the differences, the better the model fits the data.

The least squares estimates in this case are given by simple formulas:

$$\begin{aligned}\beta_1 &= \frac{\sum x_i y_i - \frac{1}{n}\sum x_i \sum y_i}{\sum x_i^2 - \frac{1}{n}(\sum x_i)^2} \\ &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\ &= \frac{\mathrm{Cov}[x, y]}{\mathrm{Var}[x]} \\ \beta_0 &= \bar{y} - \beta_1 \bar{x}\end{aligned} \tag{10.1}$$

where,

1. Var(.) and Cov(.) are called sample parameters.
2. β_1 is the slope
3. β_0 is the intercept

10.1.2 Evaluation

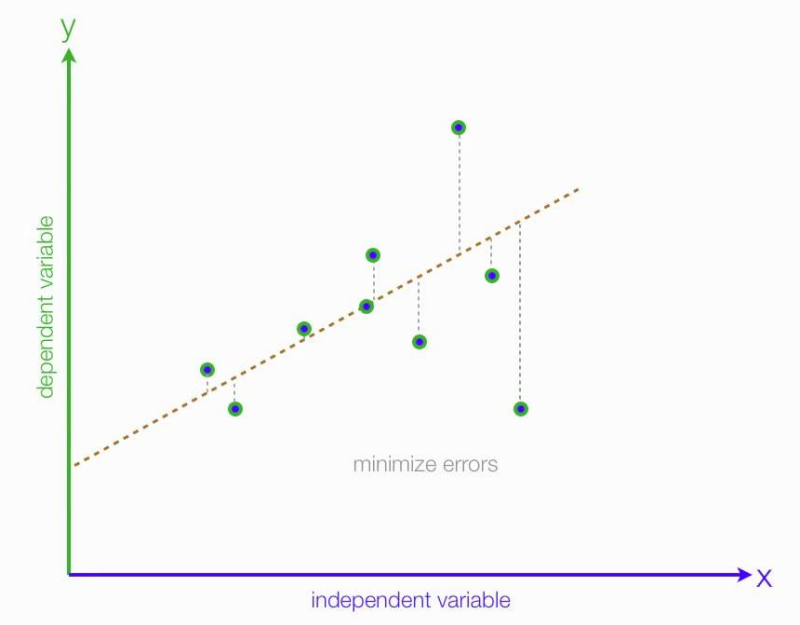
There are many methods to evaluate models. We will use Root *Mean Squared Error* and *Coefficient of Determination*

1. **Root Mean Squared Error** is the square root of sum of all errors divided by number of values:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{1}{n}(\hat{y}_i - y_i)^2} \tag{10.2}$$

2. **R^2 Error** score usually range from 0 to 1. It will also become negative if the model is completely wrong:

$$\begin{aligned} \mathrm{R}^2 &= 1 - \frac{\text{Total Sum of Squares}}{\text{Total Sum of Square of Residuals}} \\ &= 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \end{aligned} \tag{10.3}$$



Note: We want to minimize the error of our model. A good model will always have least error. We can find this line by reducing the error. The error of each point is the distance between line and that point as illustrated as above

Citations

References

1. Linear Regression
2. Ordinary Least Square

CHAPTER 11

Example Simple Linear Regression

Different methods used to demonstrate Simple Linear Regression

- Ordinary Least Square
  - Python from scratch
  - Scikit
- Gradient Descent
  - Python from scratch
  - Scikit

11.1 Ordinary Least Sqaure

Creating sample data :

```
np.random.seed(0)
X = 2.5 * np.random.randn(100) + 1.5
res = 0.5 * np.random.randn(100)
y = 2 + 0.3 * X + res
```

11.1.1 Python

Calculating model coefficiants β_0 and β_1 :

```
numer = 0
denom = 0
for i in range(100):
    numer += (X[i] - mean_x) * (y[i] - mean_y)
    denom += (X[i] - mean_x) ** 2

```

(continues on next page)

(continued from previous page)

```
b1 = numer / denom
b0 = mean_y - (b1 * mean_x)
```

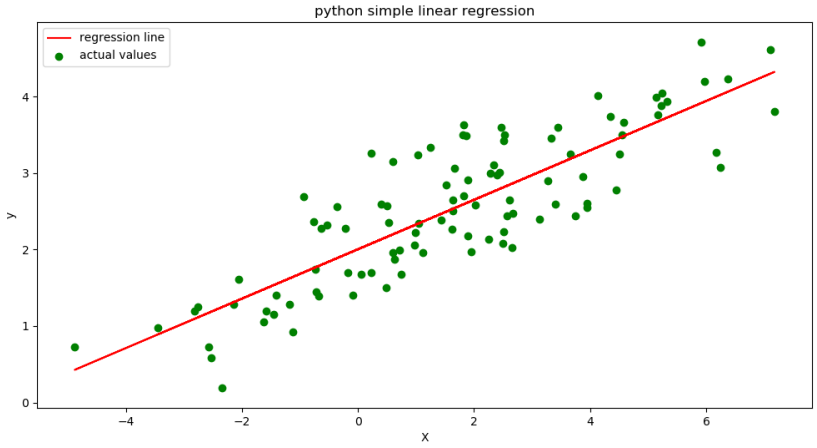
```
intercept: 2.0031670124623426
slope: 0.32293968670927636
```

Making predictions :

```
ypred = b0 + b1 * X
```

Plotting the regression line :

```
plt.figure(figsize=(12, 6))
plt.plot(X, ypred, color='red', label='regression line')      # regression line
plt.scatter(X, y, c='green', label='actual values')    # scatter plot showing actual data
```



Calculate the root mean squared error and r ^{2} error :

```
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.savefig("figure_1.png")

# calculate the error
# root mean squared
rmse = 0
for i in range(100):
    y_pred = b0 + b1 * X[i]
    rmse += (y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/100)
print("rmse: ", rmse)

# r-squared error
ss_t = 0
ss_r = 0
```

```
rmse: 0.5140943138643506
r2: 0.7147163547202338
```

11.1.2 Scikit

Reshape array, scikit cannot use array with rank 1 :

```
X = X.reshape((100, 1))
```

Initialize model and fit data :

```
reg = LinearRegression()

# Fitting training data
reg = reg.fit(X, y)
```

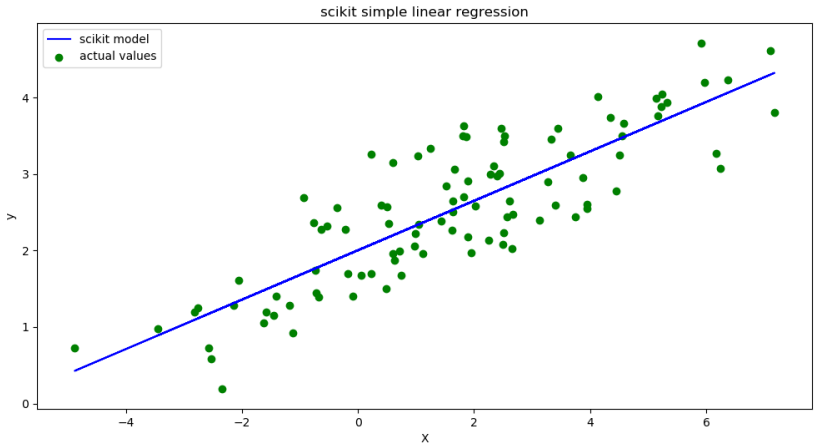
```
intercept  2.003167012462343
slope:  [0.32293969]
```

Making predictions :

```
ypred = reg.predict(X)
```

Plotting the regression line :

```
plt.clf()
plt.plot(X, ypred, color='blue', label='scikit model')
plt.scatter(X, y, c='green', label='actual values')
```



Calculate the root mean squared error and r ^{2} error :

```
# Calculating RMSE and R2 Score
mse = mean_squared_error(y, ypred)
rmse = np.sqrt(mse)
r2_score = reg.score(X, y)
print("rmse : ", rmse)
```

```
rmse :  0.5140943138643504
r2 :  0.7147163547202338
```

11.2 Gradient Descent

Creating sample data :

```
np.random.seed(33)
X = 2.5 * np.random.randn(100) + 1.5
res = 0.5 * np.random.randn(100)
Y = 2 + 0.3 * X + res
```

11.2.1 Python

Perform batch gradeint decent with `iteration=5000` and `learning_rate=0.001` :

```
for i in range(epochs):
    # The current predicted value of Y
    Y_pred = slope*X + intercpt

    # derivative wrt to slope
    D_m = (-2/n) * sum(X * (Y - Y_pred))

    # derivative wrt intercept
    D_c = (-2/n) * sum(Y - Y_pred)

    slope = slope - alpha * D_m  # Update m
    intercpt = intercpt - alpha * D_c  # Update c

    # reset for next run
    D_m = 0
    D_c = 0

    predictions.append(Y_pred)
    intercpts.append(intercpt)
    slopes.append(slope)
```

```
intercept 1.8766343225257833
slope 0.33613290217142805
```

Calculating the Errors :

```
# root mean squared
rmse = 0
for i in range(100):
    y_pred = intercpt + slope * X[i]
    rmse += (Y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/100)
# rmse = np.sqrt ( sum ( np.square ( Y - predictions[-1] ) ) ) / 100
print("rmse: ", rmse)


# r-squared error
```

(continues on next page)

(continued from previous page)

```
ss_t = 0
ss_r = 0
for i in range(100):
    y_pred = intercpt + slope * X[i]
    ss_t += (Y[i] - np.mean(Y)) ** 2
    ss_r += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_r/ss_t)
# r2 = 1 - ( sum ( np.square ( Y - np.mean(Y) ) ) / sum ( np.square ( Y - predictions[-1] ) ) )
print("r2 : ", r2)
```

```
rmse:  0.5135508150299471
r2 :  0.7398913279943503
```

Plot using `matplotlib.animation` :

```
ax.scatter(X, Y, label='data points')
line, = ax.plot([], [], 'r-', label='regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('python gradient decent')
plt.legend()
anim =  FuncAnimation(fig, update, frames=np.arange(0, 5000, 50), interval=120, repeat=False)
anim.save('figure_03.gif', fps=60, writer='imagemagic' )
```

11.2.2 Scikit

Initialize model, default `max_iteration=1000` :

```
clf = SGDRegressor(alpha=alpha)
```

Start training loop. `SGDRegressor.partial_fit` is used as it sets `max_iterations=1` of the model instance as we are already executing it in a loop. At the moment there is no callback method implemented in scikit to retrieve parameters of the training instance , therefor calling the model using `partial_fit` in a for-loop is used :

```
for counter in range (0,epochs):

    # partial fit set max_iter=1 internally
    clf.partial_fit(X, Y)
    predictions.append(clf.predict(X))
    intercpts.append(clf.intercept_)
    slopes.append(clf.coef_)
```

```
intercept 1.877731096750561
slope 0.3349640669632063
```

Note: Alternatively you could use `clf = SGDRegression(max_iter=epochs, alpha=0.001, verbose=1)` if you dont want to get the model parameters during the traing process. `verbose=1` enables `stdout` of the training process.

Calculate the Errors :

```
# Calculating RMSE and R2 Score
mse = mean_squared_error(Y, clf.predict(X))
rmse = np.sqrt(mse)
r2_score = clf.score(X, Y)
print("rmse : ", rmse)
print("r2 : ", r2_score)
```

```
rmse :  0.5135580867991415
r2 :  0.7398839617763866
```

Plot the training loop using `matplotlib.animation` :

```
ax.scatter(X, Y, label='data points')
line, = ax.plot([], [], 'r-', label='regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('scikit gradient decent')
plt.legend()
anim =  FuncAnimation(fig, update, frames=np.arange(0, epochs, 50), interval=120,␣
↪repeat=False)
anim.save('figure_04.gif', fps=60, writer='imagemagic' )
```

Important: scikits SGDRegressor converges faster than our implementation.

Citations

References

CHAPTER 12

Multiple Linear Regression

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

Multiple regression is the extension of ordinary least-squares (OLS) regression that involves more than one explanatory variable.

$$y_i = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_n * x_n \tag{12.1}$$

where

- y_i = dependent variable
- x_i = explanatory variable
- β_0 = intercept
- β_1 = slope
- ϵ = model's error term, also called residual

A simple linear regression is a function that allows to make predictions about one variable based on the information that is known about another variable. Linear regression can only be used when one has two continuous variables — an independent variable and a dependent variable. The independent variable is the parameter that is used to calculate the dependent variable or outcome. A multiple regression model extends to several explanatory variables.

The multiple regression model is based on the following assumptions:

- There is a linear relationship between the dependent variables and the independent variables.
- The independent variables are not too highly correlated with each other.
- y_i observations are selected independently and randomly from the population.
- Residuals should be normally distributed with a mean of 0 and variance σ

12.1 Least Squared Residual

12.1.1 Method

A general multiple-regression model can be written as

$$y_i = \beta_0 * 1 + \beta_1 * x_{i1} + \beta_2 * x_{i2} + \beta_k * x_{ik} + \epsilon_i \tag{12.2}$$

In matrix form, we can rewrite this model as :

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ & & \vdots & & \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix} * \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \tag{12.3}$$

The strategy in the least squared residual approach (ordinary least square) is the same as in the bivariate linear regression model. The idea of the ordinary least squares estimator (OLS) consists in choosing β_i in such a way that, the sum of squared residual (i.e. $\sum_{i=1}^{N} \epsilon_i$) in the sample is as small as possible. Mathematically this means that in order to estimate the β we have to minimize $\sum_{i=1}^{N} \epsilon_i$ which in matrix notation is nothing else than $e'e$.

$$e'e = \begin{bmatrix} e_1 & e_2 & \cdots & e_N \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} = \sum_{i=1}^{N} e_i^2$$

Consequently we can write $e'e$ as $(Y - X\beta)'(Y - X\beta)$ by simply plugging in the expression $e = Y - X\beta$ into $e'e$. This leaves us with the following minimization problem:

$$\begin{aligned} min_\beta e'e &= (Y - X\beta)'(Y - X\beta) \\ &= (Y' - \beta'X')(Y - X\beta) \\ &= Y'Y - \beta'X'Y - Y'X\beta + \beta'X'X\beta \\ &= Y'Y - 2\beta'X'Y + \beta'X'X\beta \end{aligned} \tag{12.4}$$

Note: It is important to understand that $\beta'X'Y = (\beta'X'Y)' = Y'X\beta$. As both terms are are scalars, meaning of dimension 1×1, the transposition of the term is the same term.

In order to minimize the expression in (4), we have to differentiate the expression with respect to β and set the derivative equal zero.

$$\begin{aligned} \frac{\partial(e'e)}{\partial b} &= -2X'Y + 2X'X\beta \\ -2X'Y + 2X'X\beta &\stackrel{!}{=} 0 \\ X'X\beta &= X'Y \\ \beta &= (X'X)^{-1}X'Y \end{aligned} \tag{12.5}$$

Note: the second order condition for a minimum requires that the matrix $X'X$ is positive definite. This requirement is fulfilled in case X has full rank.

Intercept

You can obtain the solution for the intercept by setting the partial derivative of the squared loss with respect to the intercept β_0 to zero. Let $\beta \in \mathbb{R}$ denote the intercept, $\beta \in \mathbb{R}^d$ the coefficients of features, and $x_i \in \mathbb{R}$ the feature vector of the i-th sample. All we do is solve for β_0 :

$$\sum_{i=1}^{n} \beta_0 = \sum_{i=1}^{n} (y_i - x_i^\top \beta)$$
$$\beta_0 = \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \beta)$$

Usually, we assume that all features are centered, i.e.,

$$\frac{1}{n} \sum_{i=1}^{n} x_{ij} = 0 \qquad \forall j \in \{1, \ldots, d\}$$

Which simplifies the solution for β_0 to be the average response :

$$\begin{aligned} \beta_0 &= \frac{1}{n} \sum_{i=1}^{n} y_i - \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{d} x_{ij} \beta_j \\ &= \frac{1}{n} \sum_{i=1}^{n} y_i - \sum_{j=1}^{d} \beta_j \frac{1}{n} \sum_{i=1}^{n} x_{ij} \\ &= \frac{1}{n} \sum_{i=1}^{n} y_i \end{aligned} \tag{12.6}$$

If in addition, we also assume that the response y is centered, i.e., $\frac{1}{n} \sum_{i=1}^{n} y_i = 0$, the intercept is zero and thus eliminated.

12.1.2 Evaluation

The coefficient of determination (R-squared) is a statistical metric that is used to measure how much of the variation in outcome can be explained by the variation in the independent variables. R^2 always increases as more predictors are added to the MLR model even though the predictors may not be related to the outcome variable.

R^2 by itself can't thus be used to identify which predictors should be included in a model and which should be excluded. R^2 can only be between 0 and 1, where 0 indicates that the outcome cannot be predicted by any of the independent variables and 1 indicates that the outcome can be predicted without error from the independent variables.

When interpreting the results of a multiple regression, beta coefficients are valid while holding all other variables constant (*all else equal*). The output from a multiple regression can be displayed horizontally as an equation, or vertically in table form.

Citations

References

1. OLS Estimator
2. Yamano Lecture Note

CHAPTER 13

Example Multiple Linear Regression

Different methods used to demonstrate Multiple Linear Regression

- Ordinary Least Square
  - Python from scratch
  - Scikit
- Gradient Descent
  - Python from scratch
  - Scikit

13.1 Ordinary Least Square

Loading Boston house-price from sklearn.datasets :

```
7  data = datasets.load_boston()
```

Define the data predictors and the target data :

```
12 # define the data/predictors as the pre-set feature names
13 df = pd.DataFrame(data.data, columns=data.feature_names)
14
15 # Put the target (housing value -- MEDV) in another DataFrame
16 target = pd.DataFrame(data.target, columns=["MEDV"])
```

13.1.1 Python

Using the Ordinary Least Square method derived in the previous section :

```
Xt = np.transpose(X)
XtX = np.dot(Xt,X)
Xty = np.dot(Xt,y)
coef_ = np.linalg.solve(XtX,Xty)
```

Set of coefficiants as calculated :

```
[   -9.28965170e-02  4.87149552e-02 -4.05997958e-03  2.85399882e+00
    -2.86843637e+00  5.92814778e+00 -7.26933458e-03 -9.68514157e-01
    1.71151128e-01 -9.39621540e-03 -3.92190926e-01  1.49056102e-02
    -4.16304471e-01]
```

13.1.2 Scikit

Lets define our regression model :

```
model = linear_model.LinearRegression(fit_intercept=False)
```

Note: we are using the same `linear_model` as in our simple linear regression method. Also the `fit_intercept` has been set to `False`. This is just to validate our derivation in the previous section. `fit_intercept` by default is set to `True` and will not assume that our response y is centered and will give an `model.intercept_` value.

Fitting our model :

```
model = model.fit(X,y)
```

```
[   -9.28965170e-02  4.87149552e-02 -4.05997958e-03  2.85399882e+00
    -2.86843637e+00  5.92814778e+00 -7.26933458e-03 -9.68514157e-01
    1.71151128e-01 -9.39621540e-03 -3.92190926e-01  1.49056102e-02
    -4.16304471e-01]
```

Evaluating our model :

```
y_predicted = model.predict(X)
print("Mean squared error: %.2f" % mean_squared_error(y, y_predicted))
print('R^2: %.2f' % r2_score(y, y_predicted))
```

```
Mean squared error: 24.17
R^2: 0.71
```

CHAPTER 14

Indices and tables

- genindex
- modindex
- search