Activation function: It takes in z = w*x and if actf(z)> threshold(theta) then 1 and -1 otherwise.

for perceptron activation function is unit step, i.e. actf(z) = 1 if z>theta and actfz(z) = -1 if z < theta

Perceptron rule: w_change = ita*(y_true - y_pred)*x

After w_change is calculated , weights are updated w_new = w_change + w_old where ita is learning rate generally between 0,1

It is important to note that the convergence of the perceptron is only guaranteed if the two classes are linearly separable and the learning rate is sufficiently small otherwise weights keep getting updated. To avoid this , run the perceptron for a fixed number of epochs.

ADALINE: Adaptive Linear Neuron

Uses identity function as activation function. actf(w_transpo*x) = w_transpo*x

and a step function for classification (quantizer)

In the case of Adaline, we can define the cost function J to learn the weights as the Sum of Squared Errors (SSE) between the calculated outcome and the true class label.

J is differentiable and convex thus "Gradient descent" a simple yet powerful optimization algo can be used to get weights minimizing the cost function.

Choosing a Classification algo

"No Free Lunch" theorem: no single classifier works best across all possible scenarios. In practice,

it is always recommended that you compare the performance of at least a handful of different learning algorithms to select the best model for the particular problem;

Once you have decided with complexity, train with all the training data so that you do not miss valuable data.

these may differ in the

1. number of features or samples,
2. the amount of noise in a dataset, and
3. whether the classes are linearly separable or not.

Steps to solve a Machine learning problem:

1. Selection of features.
2. Choosing a performance metric.
3. Choosing a classifier and optimization algorithm.
4. Evaluating the performance of the model.
5. Tuning the algorithm.

Scaling features

Use the same scaling parameters to standardize the test set so that both the values in the training and test dataset are comparable to each other. Use the same scaler that was fit using training data.

Rescaling the features such that they are on the same scale and have equal influence on the results.

X' = (X - Xmin)/(Xmax - Xmin)

Before scaling REMOVE THE OUTLIERS coz OUTLIERS WILL MESS WITH SCALING.

sklearn's minmaxscaler

Algorithms which involve two or more dimensions will be affected by feature scaling.

But since in regression features go with coefficients which take care of scale of that feature.

Also in decision trees decision boundaries are always vertical or horizontal rendering it unaffected by the size of different features.

Kmeans and SVMs will be affected by scaling coz distance calculation is involved with different ### dimensions.

Logistic Regression

It is a classification algorithm despite the name.

A linear model, used for binary classification, can be extended to multiclass classificatioin using OVR(one vs rest) technique.

Probabalistic model:

Odds ratio = p/(1-p) , where p is probability of the even happening.

Logistic regression is used in weather forecasting, for example, to not only predict if it will rain on a particular day but also to report the chance of rain. Similarly, logistic regression can be used to predict the chance that a patient has a particular disease given certain symptoms, which is why logistic regression enjoys wide popularity in the field of medicine.

logit(p) = log(p/(1-p))

z = w_transpose . x(input)

f(z) = Logistic function = 1/(1 + e^(-z)) = Probability of event occuring = p

1/(1+e^(-z)) is called sigmoid function because of the s shape it has.

for given weights and x, z = w_transpose. x can be found and using sigmoid function and z probability is found.

Logistic regression can be used to predict probabilities and class labels.

For updating weights

Maximizing the log likelihood = Minimizing the cost function

$\Delta w = -eta. \nabla J(w)$

Naive Bayes

Bayes Rule:

P(c) = Probability of cancer(event occuring). (Prior Probability)

Test evidence: P(pos/c) = Probability that test is positive when cancer is there. -- Sensitivity (SENACPO) P(pos/-c) = Probability that test is negative when cancer is not there. -- Specificity (SPACNE)

Remember:

SENACPO -- Number of times we are correct when ACtual value is POsitive. SPACNE -- Number of times we are correct when ACtual value is NEgative.

BAYES RULE:

(PRIOR PROB) . (TEST EVIDENCE) -> (POSTERIOR PROB)

P(c) = 0.01 (1%) | P(-c) = 0.99 (99%)

SENSITIVITY: P(pos/c) = 0.9 (90%) | P(neg/c) = 0.1 (10%)

SPECIFICITY: P(neg/-c) = 0.9 (90%)| P(pos/-c) = 0.1 (10%)

JOINT PROB :

P(c/pos) = P(c) . P(pos/c) = 0.01 x .90 = 0.009

P(-c/pos) = P(-c) . P(pos/-c) = 0.99 x 0.1 = 0.099

joint Probabilities generally dont add up to 1. Normalize them to make them add upto 1.

(Normalizer)factor = P(c/pos) + P(-c/pos) = 0.108

ACTUAL POSTERIOR PROB:

P(c/pos) = 0.009/factor = 0.08333 (8.33%)

P(-c/pos) = 0.099/factor = 0.9166 (91.67%)

P(c/pos) + P(-c/pos) = 0.0833 + 0.9166 = 1.0

Uses of Naive Bayes:

It is used a lot for text learning.

Why Naive?

Ignores word order, only considers the frequency of words.

Strengths and Weaknesses

Strengths:

1. Can handle lots of words (features) 20k and more.
2. Easy to implement.

Weakness:

1. Ignores word order.

Overfitting: High Variance: Can't generalize

Overfitting is a common problem in machine learning, where a model performs well on training data but does not generalize well to unseen data (test data).

If a model suffers from overfitting, we also say that the model has a high variance, which can be caused by having too many parameters that lead to a model that is too complex given the underlying data.

Underfitting: High Bias: Highly susceptible to data.

means that our model is not complex enough to capture the pattern in the training data well and therefore also suffers from low performance on unseen data.

Variance: Very Different results for different dataset

Bias: Far from correct values.

Bias Variance Trade-off

Bias:

1. A high bias ml algo doesn't learn anything from data, practically ignores it.
2. Pays little attention to data.
3. High error on training set (low Rsquared, high SSE)
4. Oversimplified

Variance: Willingness and flexibility of an algo to learn.

1. A high variance ml algo is highly susceptive to data and can't generalize.
2. Memorizes the data.
3. Fails to generalize well.
4. Much Higher error on testing set (low Rsquared, high SSE)
5. Overfitting

In stats : Variance means spread of a data distribution.

Underfit -High bias ----> Good Model ----> Overfit High Variance

Finding the optimal number of features for the good model which balances bias and variance. Can be done by:

1. Regularization: Penalizes for extra features.

Tackling overfitting via regularization

Regularization is a very useful method to handle collinearity (high correlation among features), filter out noise from data, and eventually prevent overfitting. The concept behind regularization is to introduce additional information (bias) to penalize extreme parameter weights. The most common form of regularization is the so-called L2 regularization (sometimes also called L2 shrinkage or weight decay

$C = 1/\lambda$

where, λ = Regularization Strength = More the lambda lesser the weights. C = Inverse regularization parameter (in Logistic Regression sklearn). By tuning C we can control regularization of in Logistic Regression.

More the C more the weights.

As C decreases weights shrink.

L1 Regularization: linear sum(|w|) --> encourages sparsity, diamond shaped.

L2 Regularization: quadratic sum(w^2) --> (circular)

Regularization and Feature scaling

Regularization is another reason why feature scaling such as standardization is important. For regularization to work properly, we need to ensure that all our features are on comparable scales.

SVM - Support Vector Machines

Finding a HyPlane(plane separating classes) which is at max distance from nearest samples(support vectors) Maximizing margin between Hyperplane and support vectors(samples closest to HyPlane)

Bias-Variance can be controlled by C : Higher the C more penalty for misclassifications lesser variance and more the bias.

Lower C lesser penalties softer margins.

Kernel trick: Can be used to tap into high dimensional space to find a linear separating plane in the high dimension where the data seems non linear in regular dimension. Projecting non linear combination on features to higher dimensional space.

Roughly speaking, the term kernel can be interpreted as a similarity function between a pair of samples.

Gamma = Influence on a sample on nearby region. More the gamma less the influence.

SVMs are hard to scale,

SVM - Support Vector Machine

Finds a hyperplane/line(2d) which separates the classes being at max distance from the nearest datapoints of the classes.

The distance at which hyplane or line is from the nearest point is called MARGIN.

Best Hyperplane/line is the one which maximizes the margin from classes and has most correct classifications. Priority of SVM is correct classification then margin.

Tolerates outliers easily. Robust to outliers.

Non Linear SVMs

Adding a feature from mathematical combination of existing features e.g z = x^2 + y^2 or z = |x| finds a hyperplane where it's impossible to separate classes linearly using original features.

Uses kernels to tap high dimensional space to convert non linearly separable variables in low dimension, finds a hyperplane in high dimension and returns the solution to lower dimension in form of a non linear separator.

Parameters in SVM

Kernel = rbf, sigmoid, poly , custom, linear etc.

Gamma = radial influence of single data point low gamma meaning far influence, high gamma meaning close influence. The 'gamma' parameter actually has no effect on the 'linear' kernel for SVMs. The key parameter for this kernel function is "C".

C = Controls the tradeoff between simple decision boundary and correctly classifying training points. Larger the C more the correct classifications, lower the C simpler the decision boundary.

Overfitting can be controlled by parameters of the algo, for example in case of SVMs C, Gamma, Kernel.

Advantages

Memory efficient as uses only subset of training points. Performs well in high dimensional data.

Disadvantages

Doesn't perform well with lots and lots of data as order is n^3. Doesn't work well with lots of noise. Very slow compared to Naive Bayes.

SVM tips:

1. Changing kernel can improve accuracy drastically eg. rbf - 48% to linear - 97%.
2. Reducing sample size increases training and prediction speed but reduces testing accuracy.
3. SVMs do not scale well. (O(n) = n^2 , quadratic order)
4. Optimized rbf 99% , linear 97%

SGDClassifier can be used for efficient scaling to very large datasets.

Different algos from SGDClassifier.

from sklearn.linear_model import SGDClassifier

ppn = SGDClassifier(loss='perceptron')

lr = SGDClassifier(loss='log')

svm = SGDClassifier(loss='hinge')

http://scikit-learn.org/stable/modules/sgd.html (http://scikit-learn.org/stable/modules/sgd.html)

Decision Trees

Maximizing information gain at each node.

Commonly used splitting criteria(Measures of impurity):

1. Gini index - Ig = 1 - $\sum p(i/t)$
2. Entropy - Ie = $-\sum p(i/t)\log_2(p(i/t)$
3. Classification error - Ie = 1 - max(p(i/t))

Here, (p(i/t) is the proportion of the samples that belongs to class c for a particular node t.

in practice both the Gini index and entropy typically yield very similar results and it is often not worth spending much time on evaluating trees using different impurity criteria rather than experimenting with different pruning cut-offs.

Classification error is useful for pruning but not for growing tree.

Decision Trees

creates linear decision boundaries.

Parameters

min_samples_split = 2(Default) Means won't split if samples at a node < min_sample_split

More the min_samples_split, lesser the splits, lesser the complexity, lesser the overfitting.

Entropy -

Measure of impurity in a bunch of examples.

Purity: Having all examples of the same class in a splitted section.

Entropy/impurity: Having more than 1 examples of other class at a node.

Entropy is defined for a node. A node might have multiple classes and thus entropy if a node has only one class, it is a pure node and entropy is 0.

Entropy = 1.0 when examples are evenly split amongst classes. Entropy = 0 when only one class is present in a split. Pure !

Objective: Minimizing impurity in splitting.

Entropy = $-\sum_i (p_i) \log_2 (p_i)$

where, i is a class and pi is % of that class in the split.

Information Gain

Gain = entropy(parent) - [weighted average]entropy(children)

[weighted average] is calculated basis proportion of samples going in a split. exam 2/3 and 1/3

More Gain, lesser entropy in children, more purity, better classification. Objective: Maximize Gain. Decision trees maximize gain.

Gini and Entropy:

sklearn has two criterion namely gini and entropy. Default is gini.

Decision Tree :

Strengths: Can make bigger classifiers(Ensembled Methods).

Weakness: Overfitting. (Be careful about parameter tuning).

Reducing complexity of algorithms and improving speed

1. Tune parameters
2. Identify necessary features and only use them for building model. (Generally more features the algo has, the more complex it is for fitting)

In [1]:
```
# Entropy calculator
from math import log2
def entrocalc(class_samples):
    entropy = 0
    tot_samples = 0
    for val in class_samples:
        tot_samples += class_samples[val]

    for key in class_samples:
        pi = class_samples[key]/(tot_samples)
        entropy = entropy - (pi)*log2(pi)
    return entropy
```

Random Forests - Combining weak learners to Strong ones

1. Randomly choose n samples from training set.
2. Grow a decision tree select d random features at each node.
3. Repeat steps 1 and 2 , k times.
4. Aggregate the prediction by each tree to assign a class label through majority vote.

A resonable default for d is sqrt(m) where m is nof features in training set.

The idea behind ensemble learning is to combine weak learners to build a more robust model, a strong learner, that has a better generalization error and is less susceptible to overfitting.

a big advantage of random forests is that we don't have to worry so much about choosing good hyperparameter values. We typically don't need to prune the random forest since the ensemble model is quite robust to noise from the individual decision trees. The only parameter that we really need to care about in practice is the number of trees k (step 3) that we choose for the random forest. Typically, the larger the number of trees, the better the performance of the random forest classifier at the expense of an increased computational cost. No need to standardize or normalize tree based models.

K nearest neighbors

Finds k closest neighbors using a distance metric e.g euclidean distance, manhattan distance and assigns class label by majority vote. KNN learns training data. Simple but requires lot of memory and not scalable. Complexity grows linearly.

The curse of dimensionality

It is important to mention that KNN is very susceptible to overfitting due to the curse of dimensionality. The curse of dimensionality describes the phenomenon where the feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset. Intuitively, we can think of even the closest neighbors being too far away in a high-dimensional space to give a good estimate.

Summary of classification algorithms

We have seen that decision trees are particularly attractive if we care about interpretability. Logistic regression is not only a useful model for online learning via stochastic gradient descent, but also allows us to predict the probability of a particular event. Although support vector machines are powerful linear models that can be extended to nonlinear problems via the kernel trick, they have many parameters that have to be tuned in order to make good predictions. In contrast, ensemble methods such as random forests don't require much parameter tuning and don't overfit so easily as decision trees, which makes it an attractive model for many practical problem domains. The K-nearest neighbor classifier offers an alternative approach to classification via lazy learning that allows us to make predictions without any model training but with a more computationally expensive prediction step.

More Data Better Results

(Generally) better than even a super optimized algo.

Always visualize your data.

Outliers

Rare data points which don't follow the trend.

Causes:

1. Sensor Malfunction - to be ignored
2. Data entry errors - to be ignored
3. Freak events: - to be paid attention to. e.g. Fraud detection

Removal:

1. Train > 2.Remove(10% with max residual error) > 3.Train again

(Repeat steps 2 & three until satisfied)

Visualization is one of the most powerful tools for finding outliers!

First thing to do is to Identify and Clean the outliers

Types of Data

1. Numerical: Numbers like 234, 453.0 etc ex. age, height, score.
2. Categorical: Discrete values like gender, color, material, job title etc
3. TimeSeries: Temporal data (timestamp)
4. Text: Words

Be very careful about introducing features that come from different sources depending on the class! It’s a classic way to accidentally introduce biases and mistakes.

Data preparation

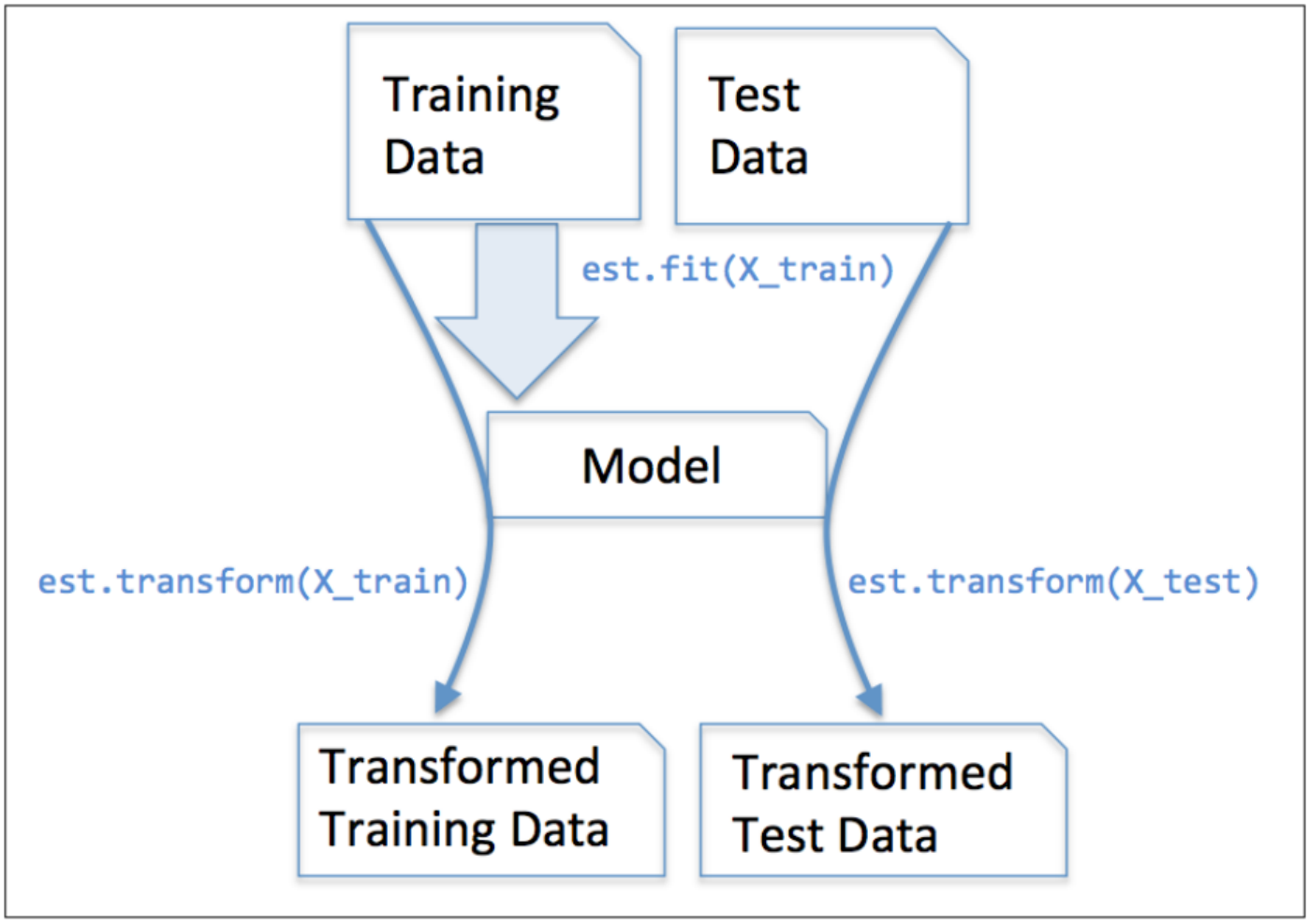
The quality of the data and the amount of useful information that it contains are key factors that determine how well a machine learning algorithm can learn. Therefore, it is absolutely critical that we make sure to examine and preprocess a dataset before we feed it to a learning algorithm.

Although scikit-learn was developed for working with NumPy arrays, it can sometimes be more convenient to preprocess data using pandas' DataFrame. We can always access the underlying NumPy array of the DataFrame via the values attribute before we feed it into a scikit-learn estimator:

Understanding sklearn estimator

Any data array that is to be transformed needs to have the same number of features as the data array that was used to fit the model.

The fit method is used to learn the parameters from the training data, and the transform method uses those parameters to transform the data.



Handling missing values

1. dropping
2. Imputing

Handling categorical data

1. LabelEncoding class labels.
2. Mapping nominal features to numbers
3. One hot encoding nominal features - Getting dummies

Splitting data into training and testing sets

Bringing features onto the same scale.

Normalization: bringing data between [0,1]

Standardization: standardization maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling, which scales the data to a limited range of values.

Overfitting

occurs when our model performs very well on training set but fails to genralize well on testing sets. It means our model is too complex for the given data. Overfitting can be reduced by:

1. Using less but important features
2. Collecting more training data
3. Introduce a penalty for complexity via regularization (explained above)
4. Reduce dimensionality of the data

Dimensionality Reduction

There are two main categories of dimensionality reduction techniques: feature selection and feature extraction.

Feature selection: we select a subset of the original features. Feature extraction: we derive information from the feature set to construct a new feature subspace.

In [ ]:

In [ ]:

In [ ]:

Sequential Feature Selection

Feature selection: selecting a subset of original features.

The motivation behind feature selection algorithms is to automatically select a subset of features that are most relevant to the problem to improve computational efficiency or reduce the generalization error of the model by removing irrelevant features or noise, which can be useful for algorithms that don't support regularization.

Sequential Backward Selection(SBS)

Removing features in stages, at each stage a feature that causes minimal accuracy loss (accuracy_before - accuracy_after) is removed.

Feature removal throught regularization (l1 and l2)

Penalizing weights.

Feature Importances

Random forest feature importance

feature importance as the averaged impurity decrease computed from all decision trees in the forest without making any assumptions whether our data is linearly separable or not. we don't need to use standardized or normalized tree-based models

scikit-learn also implements a transform method that selects features based on a user-specified threshold after model fitting, which is useful if we want to use the RandomForestClassifier as a feature selector and intermediate step in a scikit-learn pipeline, which allows us to connect different preprocessing steps with an estimator, as we will see in Chapter 6, Learning Best Practices for Model Evaluation and Hyperparameter Tuning. For example, we could set the threshold to 0.15 to reduce the dataset to the 3 most important features

Dimensionality reduction

Feature extraction

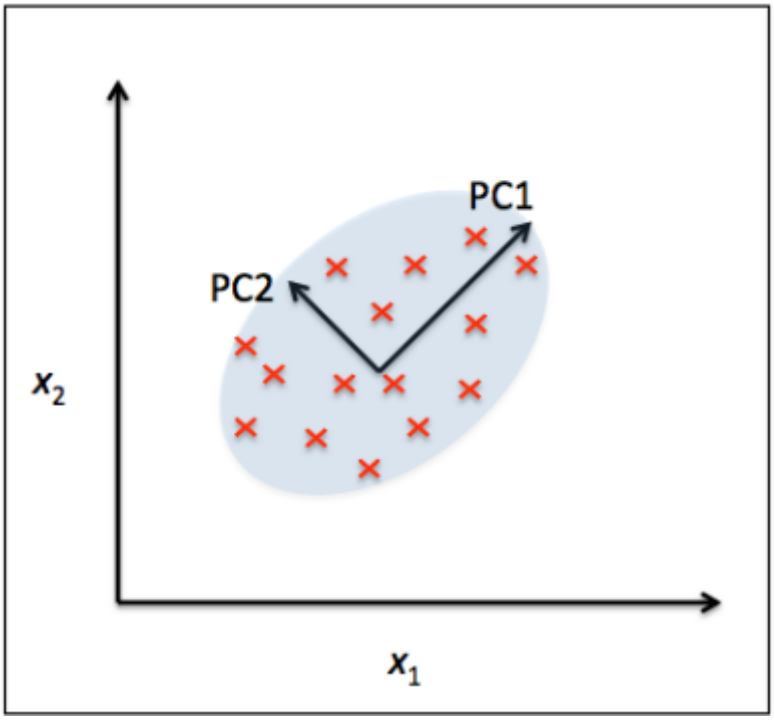
PCA: Principal component analysis Data is transformed into a low/equal dimensional feature subspace such that principal components(directions of maximum variance) are orthogonal to each other.

PCA can be used for:

1. Feature extraction(unsupervised learning)
2. De noising of signals
3. Exploratory data analysis
4. Analysis of genome data and gene expression

PCA helps us to identify patterns in data based on the correlation between features

PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions that the original one. The orthogonal axes (principal components) of the new subspace can be interpreted as the directions of maximum variance given the constraint that the new feature axes are orthogonal to each other.



Standardize features before PCA

Note that the PCA directions are highly sensitive to data scaling, and we need to standardize the features prior to PCA if the features were measured on different scales and we want to assign equal importance to all features.

Steps involved in PCA

1. Standardize the d -dimensional dataset.
2. Construct the covariance matrix. (The symmetric d d× -dimensional covariance matrix, where d is the number of dimensions in the dataset, stores the pairwise covariances between the different features.)
3. Decompose the covariance matrix into its eigenvectors and eigenvalues.
4. Select k eigenvectors that correspond to the k largest eigenvalues, where k is the dimensionality of the new feature subspace (k d≤).
5. Construct a projection matrix W from the "top" k eigenvectors.
6. Transform the d -dimensional input dataset X using the projection matrix W to obtain the new k -dimensional feature subspace

Covariance

A positive covariance between two features indicates that the features increase or decrease together, whereas a negative covariance indicates that the features vary in opposite directions.

The eigenvectors of the covariance matrix represent the principal components (the directions of maximum variance), whereas the corresponding eigenvalues will define their magnitude.

Eigen values and Eigen vectors

From a NxN covariance matrix we get N eigen vectors, eigen values are their magnitudes. We only select the subset of the eigenvectors (principal components) that contains most of the information (variance).

We are interested in the top k eigenvectors based on the values of their corresponding eigenvalues.

The variance explained ratio of an eigenvalue λ is simply the fraction of an eigenvalue λ and the total sum of the eigenvalues.

PCA is an unsupervised method, which means that information about the class labels is ignored.

PCA

principal component analysis

Finds a new coordinate system by shift-rotation of current one to reduce dimensionality.

New center is the middle point of old data range and principal axis is the one having significant variation.

Gives importance vectors \ Gives spread

art of the beauty of PCA is that the data doesn't have to be perfectly 1D in order to find the principal axis!

Making composite features using PCA to dimension reduction.

In stats : Variance means spread of a data distribution.

Principal component direction is the one that has maximum variance(spread). Because only in that direction, information loss is minimized.

More the distance of data point from principal component more the information loss.

PCA transforms features into principal components.

Principal components are used as new features.

Principal components are perpendicular to each other thus are independent.

Max nof PCs = Nof features

When to use PCA.

1. Identifying latent features driving the patterns in the data.
2. Dimensionality Reduction.

    a. Visualizing High Dimensional data.

    b. Reduce Noise

    c. Make algos work better with fewer inputs.

Higher F1 score better classifier. But more pcs don't mean better classifier, there is an optimal nof pcs that give best results.

Do not perform feature selection before PCA coz it'll throw information away. Feature selection

can be performed after PCA to help improve model.

LDA - Linear Discriminant Analysis - supervised dimensionality reduction

the goal in LDA is to find the feature subspace that optimizes class separability.

LDA AND PCA

Both LDA and PCA are linear transformation techniques that can be used to reduce the number of dimensions in a dataset; the former is an unsupervised algorithm, whereas the latter is supervised. Thus, we might intuitively think that LDA is a superior feature extraction technique for classification tasks compared to PCA. However, A.M. Martinez reported that preprocessing via PCA tends to result in better classification results in an image recognition task in certain cases, for instance, if each class consists of only a small number of samples

LDA and PCA are both linear transformation techniques. For real world non linear data transformation,

Kernel PCA for non linear mappings

we can tackle nonlinear problems by projecting them onto a new feature space of higher dimensionality where the classes become linearly separable

via kernel PCA we perform a nonlinear mapping that transforms the data onto a higher-dimensional space and use standard PCA in this higher-dimensional space to project the data back onto a lower-dimensional space where the samples can be separated by a linear classifier (under the condition that the samples can be separated by density in the input space).

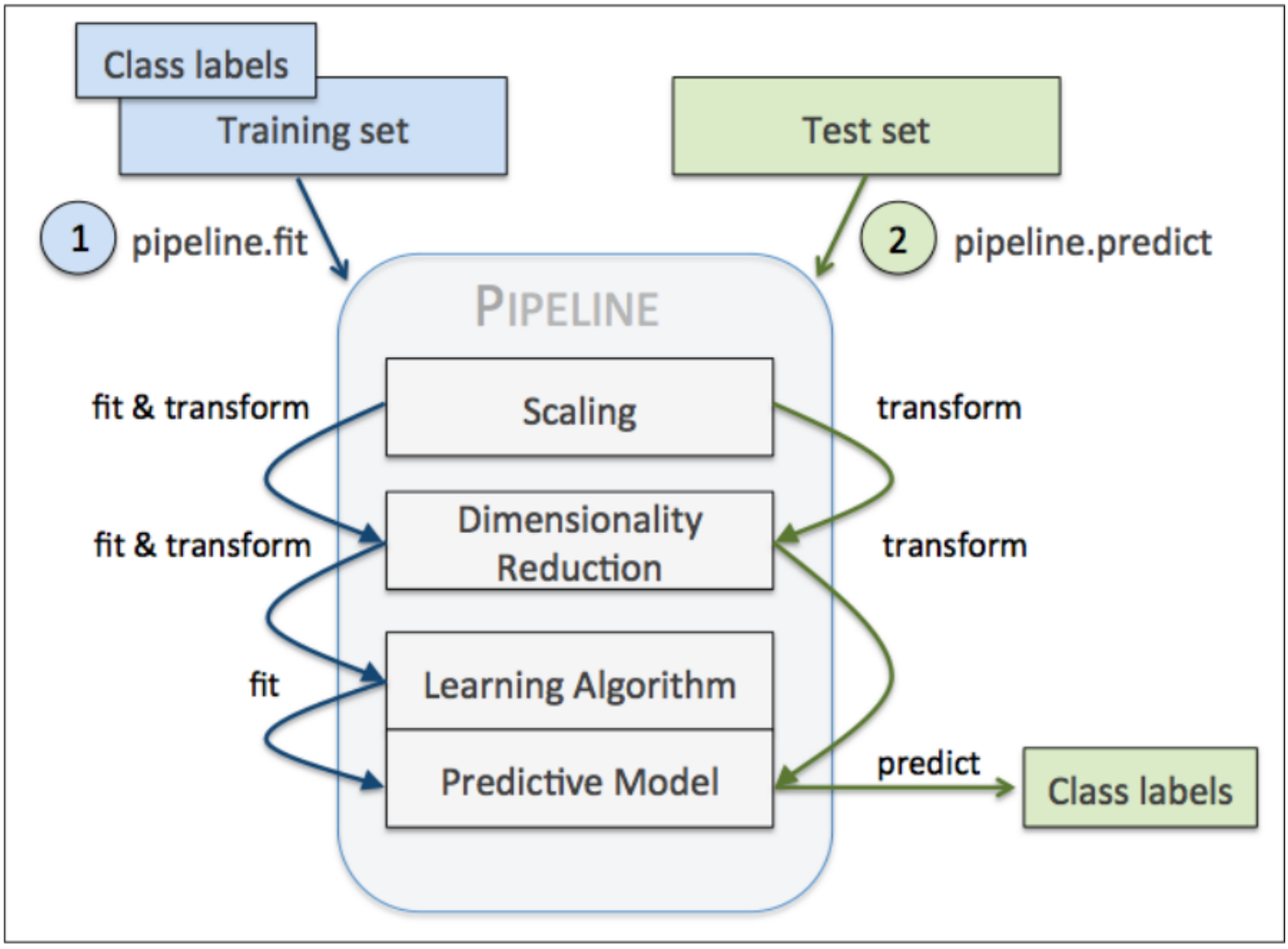
one downside of this approach is that it is computationally very expensive, and this is where we use the kernel trick. Using the kernel trick, we can compute the similarity between two high-dimension feature vectors in the original feature space.

what we obtain after kernel PCA are the samples already projected onto the respective components rather than constructing a transformation matrix as in the standard PCA approach. Basically, the kernel function (or simply kernel) can be understood as a function that calculates a dot product between two vectors—a measure of similarity

Pipeline

The Pipeline object takes a list of tuples as input, where the first value in each tuple is an arbitrary identifier string that we can use to access the individual elements in the pipeline, and the second element in every tuple is a scikit-learn transformer or estimator.

The intermediate steps in a pipeline constitute scikit-learn transformers, and the last step is an estimator.



Validation

obtain reliable estimates of the model's generalization error, that is, how well the model performs on unseen data

Holdout

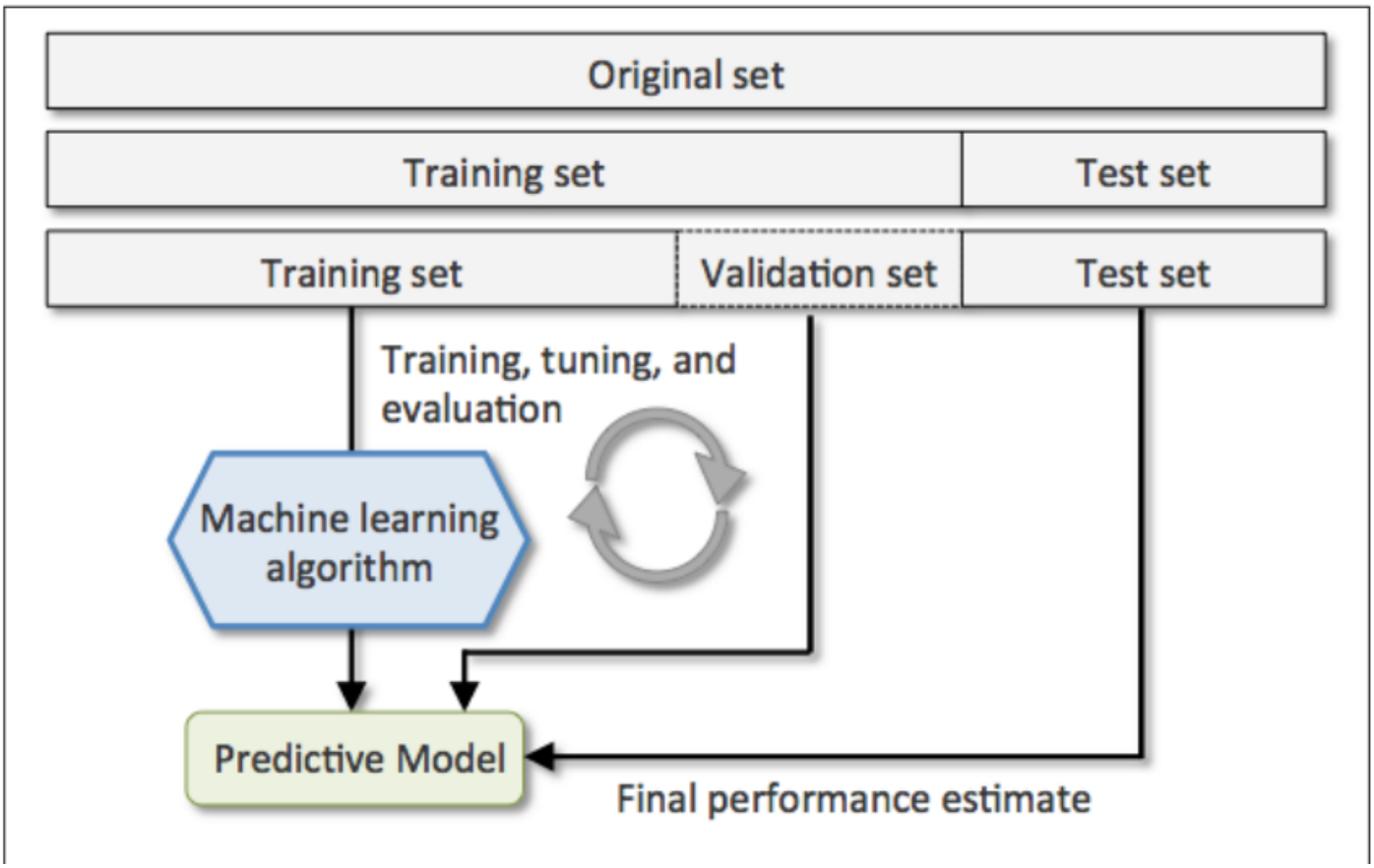
Data -> Train set, Test set

Train set -> Train set, Validation set

Training is done on training set, parameter tuning using validation set

and performance evaluation using test set.

A disadvantage of the holdout method is that the performance estimate is sensitive to how we partition the training set into the training and validation subsets; the estimate will vary for different samples of the data



Validation

Train Test split:

splitting data into training and testing sets and using only training set for training and testing set to evaluate the model.

1. Serves as a check on overfitting.
2. Gives an estimate of performance on independent set.

Flow for split,pca, model training and prediction

K-fold Cross Validation

In k-fold cross-validation, we randomly split the training dataset into k folds without replacement, where k-1 folds are used for the model training and one fold is used for testing. This procedure is repeated k times so that we obtain k models and performance estimates.

We then calculate the average performance of the models based on the different, independent folds to obtain a performance estimate that is less sensitive to the subpartitioning of the training data compared to the holdout method

Since k-fold cross-validation is a resampling technique without replacement, the advantage of this approach is that each sample point will be part of a training and test dataset exactly once, which yields a lower-variance estimate of the model performance than the holdout method.

K fold cv tips: (default number of folds: 10)

1. Small training data: Increase the number of folds
2. Large training data: Decrease the number of folds

Remember Large values of K Lower bias and Higher Variance Small values of K Higher bias and Lower Variance.

Leave one out(LOO) used for very small datasets.

Stratified K-fold

stratified k-fold cross-validation can yield better bias and variance estimates, especially in cases of unequal class proportions.

In stratified cross-validation, the class proportions are preserved in each fold to ensure that each fold is representative of the class proportions in the training dataset.

Kfold

Dividing the dataset into k subsets, taking each subset as a testing set once and remaining as training set and reporting the average of performance on K subsets.

1. Slower to train than train/test split.
2. Better estimate of model accuracy than train/test split.

Just splits the data irrespective of classes coming in the train/test. This might result into training the model on one class and using it to predict the other which will as we expect perform poorly.

Training data should be such that it has a similar presence of all the classes as in the complete data set.

Stratified K-fold ensures that

each set contains approximately the same percentage of samples of each target class as the complete set.

Learning and Validation Curves

If a model is too complex for a given training dataset—there are too many degrees of freedom or parameters in this model—the model tends to overfit the training data and does not generalize well to unseen data. Often, it can help to collect more training samples to reduce the degree of overfitting. However, in practice, it can often be very expensive or simply not feasible to collect more data.

High Bias model: low training accuracy, low cross-validation accuracy Steps to reduce bias(underfitting) {Basically try to make a more complex model}:

1. Add more features(create, add)
2. Decrease degree of regularization

High Variance model: High training accuracy, low cross-validation accuracy Steps to reduce Variance(overfitting) {Basically try to make a simpler model}:

1. If possible collect more data. (Be careful with more data, more noise might also come)
2. Increase regularization strength.
3. Decrease number of features via feature selection/extraction.

Learning Curve: Plotting training and validation accuracies for different training set sizes. Validation Curve: Plotting training and validation accuracies for different values of model parameters.

Ensemble Learning

The goal behind ensemble methods is to combine different classifiers into a meta-classifier that has a better generalization performance than each individual classifier alone.

Majority and Plurality

Majority voting simply means that we select the class label that has been predicted by the majority of classifiers, that is, received more than 50 percent of the votes. Strictly speaking, the term majority vote refers to binary class settings only. However, it is easy to generalize the majority voting principle to multi-class settings, which is called plurality voting. Here, we select the class label that received the most votes (mode).

Building Ensembles

Depending on the technique, the ensemble can be built from different classification algorithms, for example, decision trees, support vector machines, logistic regression classifiers, and so on.

Alternatively, we can also use the same base classification algorithm fitting different subsets of the training set. One prominent example of this approach would be the random forest algorithm, which combines different decision tree classifiers. The following diagram illustrates the concept of a general ensemble approach using majority voting.

Ensemble Error Probability

When more than half of the base models predict wrongly.

we make the assumption that all n base classifiers for a binary classification task have an equal error rate ε. Furthermore, we assume that the classifiers are independent and the error rates are not correlated. Under those assumptions, we can simply express the error probability of an ensemble of base classifiers as a probability mass function of a binomial distribution:

$$P\left(y \geq k\right) = \sum_{k}^{n} \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle \varepsilon^{k} \left(1-\varepsilon\right)^{n-k} = \varepsilon_{ensemble}$$

Learning:

if base error is high(0.40), more classifiers(250) would be required to build a good ensemble model.

If base error is low(0.25), less classifiers(40) would be required to build a good ensemble model.

Ensemble error is more than base error if base model performs worse than random guessing i.e 0.5 base error.

it is a bad practice to use the test dataset more than once for model evaluation,

*Using it again and improving model according results makes test data part of the training data and then using the same test data to get an idea of generalization accuracy of the model will give false inflated results.

Bagging

Sampling done with replacement to get different training sets.

A base algo trained on different training sets giving different models. Voting used to get the prediction.

Training set T: (1,2,3,4)

T1: (1,3,3,4) -> Classifier 1,C1

T2: (2,2,2,3) -> Classifier 2,C2

.

.

etc

Bagging does not reduce bias. Bagging reduces variance.

Adaptive Boosting: Weak Learners to a Strong model

Weak Learners

The key concept behind boosting is to focus on training samples that are hard to classify, that is, to let the weak learners subsequently learn from misclassified training samples to improve the performance of the ensemble. In contrast to bagging, the initial formulation of boosting, the algorithm uses random subsets of training samples drawn from the training dataset without replacement. The original boosting procedure is summarized in four key steps as follows:

Boosting rounds

First round:

1.Uniform weights initialized.

2.error calculated(true, predicted)

3.coefficient alpha calculated

4.weights modified

5.weights normalized

Next round from step 2

Adaptive boosting and boosting in general add variance but help reducing bias.

Sentiment Analysis

Bag of words

Converting text to numerical data.

```
      word1 word2 word3 word4.....wordn
text1  1     1    0      4         3
text2  0     2    1      0         1
```

Since the unique words in each document represent only a small subset of all the words in the bag-of-words vocabulary, the feature vectors will consist of mostly zeros, which is why we call them sparse

N grams

• 1-gram: "the", "sun", "is", "shining"

• 2-gram: "the sun", "sun is", "is shining"

the contiguous sequences of items in NLP—words, letters, or symbols—is also called an n-gram. The choice of the number n in the n-gram model depends on the particular application; for example, a study by Kanaris et al. revealed that n-grams of size 3 and 4 yield good performances in anti-spam filtering of e-mail messages

The CountVectorizer class in scikit-learn allows us to use different n-gram models via its ngram_range parameter. While a 1-gram representation is used by default, we could switch to a 2-gram representation by initializing a new CountVectorizer instance with ngram_range=(2,2).

term frequencies(tf): Number of times term t occurs in document d. tf(t, d)

inverse document frequency(idf):

$$\mathrm{idf}\left(\mathrm{t,d}\right) = log\,\frac{n_d}{1+\mathrm{df}\left(\mathrm{d,t}\right)},$$

nd = total number of documents

df(d,t) = number of documents that contain term t

$$\text{tf-idf}\left(\text{t,d}\right) = tf\left(t,d\right) \times \text{idf}\left(\text{t,d}\right)$$

Sklearns TfidfTransformer applies L2 normalization after calculating tfidfs

Processing document into tokens

```
Splitting: splitting text by whitespaces
Word Stemming: transforming words to their roots
```

Training heavy data sets takes a lot of time on a local machine.

Out of core learning helps tackle large datasets

we can't use the CountVectorizer for out-of-core learning since it requires holding the complete vocabulary in memory. Also, the TfidfVectorizer needs to keep the all feature vectors of the training dataset in memory to calculate the inverse document frequencies. However, another useful vectorizer for text processing implemented in scikit-learn is HashingVectorizer. HashingVectorizer is data-independent and makes use of the Hashing trick via the 32-bit MurmurHash3 algorithm by Austin Appleby

Learning From Text

Bag of Words: Frequency counts of occuring words.

Using sklearn countvectorizer

All words are not equally important, words like the/a/an/is/etc don't tell much about what's going on and so are redundant called "STOPWORDS"

Remove stopwords before starting text analysis.

STEMMER: used to consolidate different words with same stem like repond, responsiveness etc. various stemmers in nltk eg. snowball stemmer and more.

Order of operation in Text processing

1. Stop words removal
2. Stemming
3. Bag of words

TF IDF :

Term Frequency: How many times a word occurs in a document. Inverse Document Frequency: In how many document a word occurs.

Feature Selection

1. Select best features
2. Engineer new features
3. Getting Rid of features

Engineering new Feature:

1. Use Human Intuition
2. Code up the feature
3. Visualize : See if there are trends which can be utilized by ML algos.
4. Repeat

Beware of programming bugs that might creep in while engineering new features.

1. Anyone can make mistakes--be skeptical of your results!
2. 100% accuracy should generally make you suspicious. Extraordinary claims require extraordinary proof.
3. If there's a feature that tracks your labels a little too closely, it's very likely a bug!
4. If you're sure it's not a bug, you probably don't need machine learning--you can just use that feature alone to assign labels.

Getting Rid of features

Remove the feature when:

1. It's noisy
2. It's highly correlated to other feature. (Repeating information)
3. It causes overfitting
4. slows down training/testing

General Rule

Features are not equal to information. Features attempt to access information.

Goal: Bare minimum number of features that give the most info.

Univariate Feature Selection:

Treats each feature independently and asks how much power it gives you in classifying or regressing.

sklearn:

1. SelectPercentile: X% of features that are most powerful
2. SelectKBest: selects the K features that are most powerful
3. TFIDF vectorizer max_df, min_df can also help get the right features.

Text data has lots and lots of features , feature reduction can be used. Feature reduction can be used for highly dimensional data.

A classic way to overfit an algorithm is by using lots of features and not a lot of training data

Deploying on Web

Regression Analysis

Univariate: one independent(explanatory) variable

Bivariate: two independent variables

Multivariate: Many independent variables

X = explanatory variable

y = response variable

linear regression can be understood as finding the best-fitting straight line through the sample points

This best-fitting line is also called the regression line, and the vertical lines from the regression line to the sample points are the so-called offsets or residuals—the errors of our prediction.

Exploration | Visualization

Exploratory Data Analysis (EDA) is an important and recommended first step prior to the training of a machine learning model.

1. Scatter plots, histograms

Correlation Matrix

The correlation matrix is a square matrix that contains the Pearson product-moment correlation coefficients (often abbreviated as Pearson's r), which measure the linear dependence between pairs of features.

The correlation coefficients are bounded to the range -1 and 1.

r = 1 --> Perfectly positive correlation. r = 0 --> No correlation. r = -1 --> Perfectly Negative correlation.

Pearson's correlation coefficient can simply be calculated as the covariance between two features x and y (numerator) divided by the product of their standard deviations (denominator):

$$r = \frac{\sum_{i=1}^{n}\left[\left(x^{(i)} - \mu_x\right)\left(y^{(i)} - \mu_y\right)\right]}{\sqrt{\sum_{i=1}^{n}\left(x^{(i)} - \mu_x\right)^2}\sqrt{\sum_{i=1}^{n}\left(y^{(i)} - \mu_y\right)^2}} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

To fit a linear regression model, we are interested in those features that have a high correlation with our target variable MEDV

Highest correlations of target variable MEDV are with LSTAT, RM.

Looking at plots it can be observed that LSTAT has a fairly non linear relationship with MEDV. But MEDV shows a linear relationship with RM.

Fitting a Robust Regression Model

As an alternative to throwing out outliers, we will look at a robust method of regression using the RANdom SAmple Consensus (RANSAC) algorithm, which fits a regression model to a subset of the data, the so-called inliers. Using the residual_metric parameter, we provided a callable lambda function that simply calculates the absolute vertical distances between the fitted line and the sample points. By setting the residual_threshold parameter to 5.0, we only allowed samples to be included in the inlier set if their vertical distance to the fitted line is within 5 distance units, which works well on this particular dataset. By default, scikit-learn uses the MAD estimate to select the inlier threshold, where MAD stands for the Median Absolute Deviation of the target values y. However, the choice of an appropriate value for the inlier threshold is problem-specific, which is one disadvantage of RANSAC.

Residual Plots

we can plot the residuals (the differences or vertical distances between the actual and predicted values) versus the predicted values to diagnose our regression model. Those residual plots are a commonly used graphical analysis for diagnosing regression models to detect nonlinearity and outliers, and to check if the errors are randomly distributed.

In the case of a perfect prediction, the residuals would be exactly zero, which we will probably never encounter in realistic and practical applications. However, for a good regression model, we would expect that the errors are randomly distributed and the residuals should be randomly scattered around the centerline. If we see patterns in a residual plot, it means that our model is unable to capture some explanatory information, which is leaked into the residuals as we can slightly see in our preceding residual plot. Furthermore, we can also use residual plots to detect outliers, which are represented by the points with a large deviation from the centerline

Mean Squared Error

Mean Squared Error (MSE), which is simply the average value of the SSE cost function that we minimize to fit the linear regression model. The MSE is useful to for comparing different regression models or for tuning their parameters via a grid search and cross-validation:

As we saw above MSE test is more than MSE train which means model is certainly overfitting the data.

MSE order is dependent on y. A standardized version of MSE is R^2(R squared) which is

R^2 = 1 - (SSE/SST) = 1 - (MSE/var(y))

SSE = sum((y(i) - ypred(i))^2)

SST = sum((y(i) - mean(y))^2)

Regularized Regression Models

1. Ridge regression: Quadratic Penalty | L2 penalty : add lambda*sum(w^2) to the SSE(cost function)
2. Lasso regression: Linear Penalty | L1 penalty : add lambda*sum(w) to the SSE(cost function)

Depending on the regularization strength, certain weights can become zero, which makes the LASSO also useful as a supervised feature selection technique:

1. Elastic Net: combination of ridge and lasso | add L1 + L2 terms to cost function.

As lambda increases regularization strength increases and weights shrink. Intercept term is not regularized.

Important tips and conclusion

As we can see cubic fits better than linear and quadratic. As degree increases training accuracy increases but it makes model more complex which may result into overfitting.

Non linear problems can be dealt with:

1. Polynomial regression
2. Transforming variables - log, sqrt, cube root etc
3. Decision tree and Random forests, SVMs In the context of decision tree regression, the MSE is often also referred to as within-node variance, which is why the splitting criterion is also better known as variance reduction

   A random forest usually has a better generalization performance than an individual decision tree due to randomness that helps to decrease the model variance. Other advantages of random forests are that they are less sensitive to outliers in the dataset and don't require much parameter tuning. The only parameter in random forests that we typically need to experiment with is the number of trees in the ensemble.

   The only difference is that we use the MSE criterion to grow the individual decision trees, and the predicted target variable is calculated as the average prediction over all decision trees.

Regression (Continuous output)

Minimizes sum of squared errors (actual - predicted). Finds slope and intercept for the line which minimizes sum of squared errors.

absolute error minimization not used because it can give us more than one lines.

In case of squared errors there will be only one line. also SSE is easier to implement.

Problem with SSE:

1. Adding more data increases SSE but that doesn't mean fit is bad.

This is done by:

Ordinary Least squares OLS (used in sklearn)

Linear descent

Performance Measure for Regression : R-Squared

R-Squared: "How much of change in the output is explained by the change in the input.

0.0 < R-Squared < 1.0 (Best)

Negative R-Squared is possible.*

Advantage over SSE:

1. Independent of datapoints. Higher the R-Squared, the better. Max value = 1.0

Clustering

K means clustering

Steps:

1. Assign: Randomly assign cluster centers.
2. Cluster Identification: Find points nearest to these cluster centers to identify the clusters.
3. Centroid :Find the centroid of these clusters. New cluster centers are these centroids.
4. Repeat 2 & 3 until cluster centers stop updating.

sklearn params:

n_clusters = Number of clusters we want to have. n_init = How many times it is initialized. Play with this if you see clustering getting affected by initialization.

max_iter = How many iterations in total?

Limitations :

1. Premature convergence to sub optimal values.
2. Can result into different clusters based on Initialization.

business-oriented applications of clustering include the grouping of documents, music, and movies by different topics, or finding customers that share similar interests based on common purchase behaviors as a basis for recommendation engines.

Types of clustering:

1. Prototype based: Centroid(average for continuous vars) , Medoid(most occuring point for categorical features) eg. Kmeans
2. Hierarchical
3. Density based

Kmeans:

Advantages: simple, easy to implement and commputationally efficient

Disadvantages: Have to define K apriori, wrong choice of K might result in bad clustering. Another problem with k-means is that one or more clusters can be empty(addressed in scikit learn- assigning the farthest point from centroid of empty cluster as the new centroid)

Standardization/Scaling(features have to be on the same scale) required when using Euclidean distance

Hard Clustering: Each sample asssigned to only one cluster. KMeans

Soft Clustering: A sample is assigned to one or more clusters. FCM(fuzz C means)

each sample has a probability for different clusters.

Optimum number of clusters

1. Distortion(withing cluster SSE) (Inertia attribute of KMeans): How far are the samples from centroid.

if K(nof clusters) increases distortion decreases as points are closer to centroids.

Finding the K where distortion shoots up - "Look for the elbow in plot"

Silhouette Analysis: Quality of clustering

Applicable to other clustering algorithms too.

Silhouette analysis can be used as a graphical tool to plot a measure of how tightly grouped the samples in the clusters are.

Silhouette Coefficient

$$s^{(i)} = \frac{b^{(i)} - a^{(i)}}{\max\left\{b^{(i)}, a^{(i)}\right\}}$$

ai = average distance of sample xi from the samples of the cluster | quantifies how similar a sample is to other samples of the cluster

bi = average distance of sample xi from the samples of the nearest cluster | quantifies how dissimilar a sample is from nearest cluster samples

Silhouette coefficient = 1 Ideal silhoutte | b >> a

silhouette coefficient range (-1, 1)

Evaluation Metric

True Positive: When we predicted positive and actual value is positive. True Negative: When we predicted negative and actual value is negative.

False Positive: When we predicted positive and actual value is negative. False negative: When we predicted negative and actual value is positive.

Accuracy and Error

Accuracy: Correct predictions/Total predictions

Accuracy = (TP+ TN)/(TP+TN+FP+FN)

Error = Wrong predictions/Total predictions

Error = (FP+FN)/(TP+TN+FP+FN))

Accuracy = 1 - Error

TPR and FPR

The true positive rate (TPR) and false positive rate (FPR) are performance metrics that are especially useful for imbalanced class problems:

FPR = FP/N = FP/(FP+TN)

FPR = How many times we incorrectly predicted positive when actual value was negative.

TPR = TP/P = TP/(TP+FN) = How many times we correctly predicted positive when actual value was postive.

Precision and Recall

Precision (PRE) and recall (REC) are performance metrics that are related to those true positive and true negative rates, and in fact, recall is synonymous to the true positive rate

PRE = TP/(TP+FP) , how many times we are correct when we predicted positive.

Recall = TP/(TP+FN) : Same as True positive rate.

F1-score = 2*(PRExREC)/(PRE+REC)

Remember that the positive class in scikit-learn is the class that is labeled as class 1. If we want to specify a different positive label, we can construct our own scorer via the make_scorer function, which we can then directly provide as an argument to the scoring parameter in GridSearchCV:

ROC - Receiver Operating Characteristic curves

Receiver operator characteristic (ROC) graphs are useful tools for selecting models for classification based on their performance with respect to the false positive and true positive rates, which are computed by shifting the decision threshold of the classifier.

The diagonal of an ROC graph can be interpreted as random guessing, and classification models that fall below the diagonal are considered as worse than random guessing. A perfect classifier would fall into the top-left corner of the graph with a true positive rate of 1 and a false positive rate of 0. Based on the ROC curve, we can then compute the so-called area under the curve (AUC) to characterize the performance of a classification model.

Macro average for multiclass problems

learn, a normalized or weighted variant of the macro-average is used by default. The weighted macro-average is calculated by weighting the score of each class label by the number of true instances when calculating the average. The weighted macro-average is useful if we are dealing with class imbalances, that is, different numbers of instances for each label.

In [ ]: