Text Mining

Emanuele Guidotti

2019/2020

Classification

Given a set of classes, classification algorithms are **supervised learning algorithms** aimed at **assigning the correct class label** to the given input.

Classification analysis is broadly used in applications such as...

- topic identification
- spam detection
- sentiment analysis
- intent detection

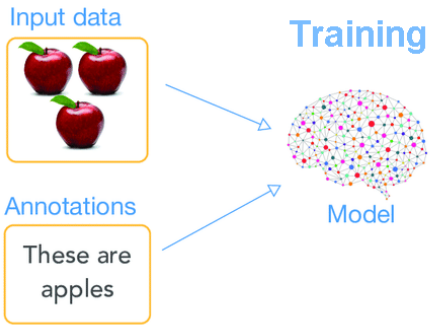
... and in every task of assigning a set of predefined categories to the given input.

Supervised Learning

Supervised Learning

Learning a function that maps an input to an output based on example input-output pairs.

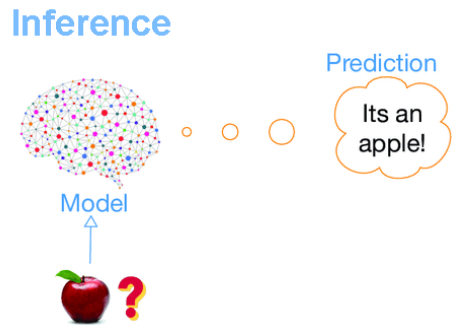
- infer a function from labeled training data
- use the inferred function to label new instances



Supervised Learning

Learning a function that maps an input to an output based on example input-output pairs.

- infer a function from labeled training data
- use the inferred function to label new instances

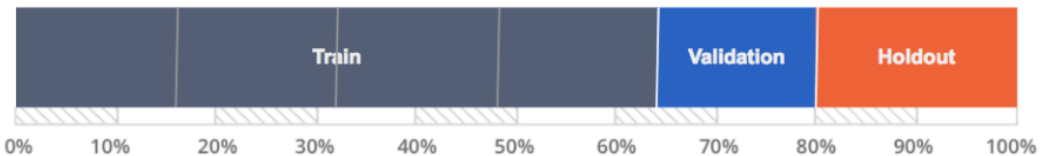


Pipeline

Step	Task	Data
Pre-processing	Cleaning data and extracting features	Fit on the training set and apply to the whole dataset
Training	Tuning model parameters	Training set
Validation	Selecting the best model	Validation set
Inference	Evaluating the final model	Holdout set

Partitioning Data

Partitioning data into training, validation, and holdout sets allows to develop highly accurate models that are relevant to data collected in the future, not just the data the model was trained on.



Training Set

A training set is the subsection of a dataset from which the machine learning algorithm uncovers, or "learns," relationships between the features and the target variable.

Validation Set

A validation set is the subset of the data to which we apply the machine learning algorithm to see how accurately it identifies relationships between the known outcomes for the target variable and the input features. It can be used to make decisions about which algorithms to use or for improving or tuning algorithms.

Holdout Set

Sometimes referred to as "testing" data, a holdout subset provides a final estimate of the machine learning model's performance after it has been trained and validated. Holdout sets should never be used to make decisions about which algorithms to use or for improving or tuning algorithms.

Feature Selection

Feature Selection

In text classification, the feature selection is the process of selecting a specific subset of the terms of the training set and using only them in the classification algorithm.

It is especially important in *text* classification due to the high dimensionality of text features and noise.

In the case of the classification problem, it makes sense to **supervise the feature selection process** with the use of the class labels. This kind of selection process ensures that those terms which are highly skewed towards the presence of a particular class label are picked for the learning process.

How to determine the most informative terms?

Gini Index

Let $p(c \mid t)$ be the conditional probability that a document belongs to class c, given the fact that it contains the term t. Therefore, we have:

$$\sum_{c=1}^{k} p(c \mid t) = 1$$

Then, the gini-index for the term t, denoted by $G(t)$ is defined as:

$$G(t) \equiv \sum_{c=1}^{k} p(c \mid t)^2$$

Gini Index

- The value of the gini-index lies in the range $(1/k, 1)$.
- Higher values of the gini-index indicate a greater discriminative power of the term t.
- If the global class distribution is skewed, the gini-index may not accurately reflect the discriminative power of the underlying attributes.
- It is possible to construct a **normalized gini-index** in order to reflect more accurately class-discrimination in the case of biased class distributions in the whole document collection.

Normalized Gini Index

Let $p(c)$ represent the unconditional probability of class c. Then, we determine the normalized probability value $p'(c \mid t)$ as:

$$p'(c \mid t) \equiv \frac{p(c \mid t)/p(c)}{\sum_{i=1}^{k} p(i \mid t)/p(i)}$$

Then, the gini-index is computed in terms of these normalized probability values.

$$G(t) \equiv \sum_{c=1}^{k} p'(c \mid t)^2$$

Mutual Information

The pointwise mutual information $M_c(t)$ between the term t and the class c is defined on the basis of the level of co-occurrence between the class c and term t.

Let $p(c)$ be the unconditional probability of class c, and $p(c \mid t)$ be the probability of class c, given that the document contains the term t. Let $p(t)$ be the fraction of the documents containing the term t, i.e. the unconditional probability of term t.

The expected co-occurrence of class c and term t on the basis of mutual independence is given by $p(c) \cdot p(t)$. The true co-occurrence is of course given by $p(c \mid t) \cdot p(t)$.

Mutual Information

In practice, the value of $p(c \mid t) \cdot p(t)$ may be much larger or smaller than $p(c) \cdot p(t)$, depending upon the level of correlation between the class c and term t. The mutual information is defined in terms of the ratio between these two values.

$$M_c(t) = log\Big(\frac{p(c \mid t) \cdot p(t)}{p(c) \cdot p(t)}\Big) = log\Big(\frac{p(c \mid t)}{p(c)}\Big)$$

Clearly, the term t is positively correlated to the class c, when $M_c(t) > 0$, and the term t is negatively correlated to class c, when $M_c(t) < 0$.

Mutual Information

Note that $M_c(t)$ is specific to a particular class c. We need to compute the overall mutual information as a function of the mutual information of the term t with the different classes.

$$M_{avg}(t) = \sum_{c=1}^{k} p(c) \cdot M_c(t)$$

$$M_{max}(t) = \max_{c}\{M_c(t)\}$$

The second measure is particularly useful, when it is more important to determine high levels of positive correlation of the term t with any of the classes.

χ^2-Statistic

The χ^2-statistic is a different way to compute the lack of independence between the term t and a particular class c. Let n be the total number of documents, then:

$$\chi^2_c(t) = \frac{n \cdot p(t)^2 \cdot (p(c \mid t) - p(c))^2}{p(t) \cdot (1 - p(t)) \cdot p(c) \cdot (1 - p(c))}$$

As in the case of the mutual information, we can compute a global χ^2 statistic from the class-specific values.

One major advantage of the χ^2-statistic is that it is a normalized value and we can test statistical significance using the χ^2 distribution with one degree of freedom.

Classification Algorithms | Rocchio Classifier

Rocchio Classifier

Each class is represented by its centroid, with test samples classified to the class with the nearest centroid.

Using a training set of documents, the Rocchio algorithm builds a prototype vector, *centroid*, for each class. This prototype is an average vector over the training documents' vectors that belong to a certain class.

$$\boldsymbol{\mu_c} = \frac{1}{|D_c|} \sum_{\boldsymbol{d} \in D_c} \boldsymbol{d}$$

Where D_c is the set of documents in the corpus that belongs to class c and $\boldsymbol{d}$ is the vector representation of document d.

The predicted label of document $\boldsymbol{d}$ is the one with the smallest (Euclidean) distance between the document and the centroid.

$$\hat{c} = arg \min_{c} ||\boldsymbol{\mu_c} - \boldsymbol{d}||$$

Vectors can be normalized to unit-length and a custom metric can be used when calculating distance between instances.

```
# class
sklearn.neighbors.NearestCentroid
```

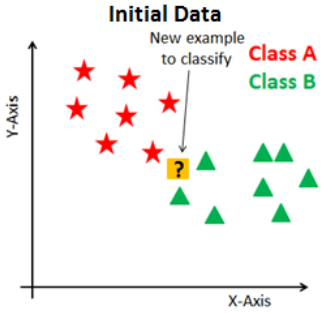
Classification Algorithms | K-Nearest Neighbor

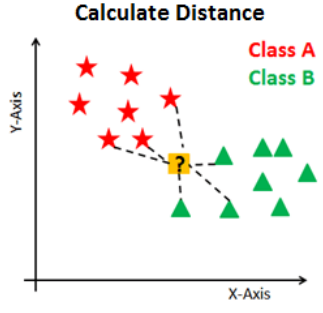
K-Nearest Neighbor

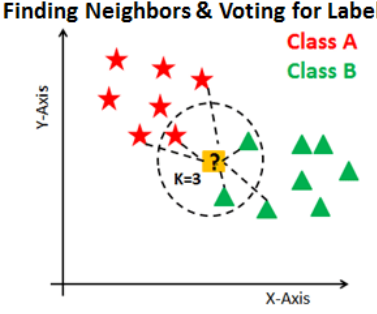
Given a test document $\boldsymbol{d}$, the KNN algorithm finds the k nearest neighbors of $\boldsymbol{d}$ among all the documents in the training set, and scores the category candidates based on the class of the k neighbors. After sorting the score values, the algorithm assigns the candidate to the class with the highest score.

The basic nearest neighbors classification uses uniform weights: that is, the value assigned to a query point is computed from a simple majority vote of the nearest neighbors. Under some circumstances, it is better to weight the neighbors such that nearer neighbors contribute more to the fit.

```
# class
sklearn.neighbors.KNeighborsClassifier
```


Initial Data
New example
to classify
Class A
Class B
Y-Axis
X-Axis
?


Calculate Distance
Class A
Class B
Y-Axis
X-Axis
?


Finding Neighbors & Voting for Labels
Class A
Class B
Y-Axis
X-Axis
?
K=3

Classification Algorithms | Naive Bayes

Naive Bayes Classifier

The Naive Bayes algorithm classifies instances with the label that is most likely given the corresponding set of features.

$$\hat{c} = arg\max_c p(c \mid \boldsymbol{d}) \overset{bayes}{=} arg\max_c \frac{p(c) \cdot p(\boldsymbol{d} \mid c)}{p(\boldsymbol{d})} =$$

$$= arg\max_c p(c) \cdot p(\boldsymbol{d} \mid c) \overset{naive}{=} arg\max_c p(c) \cdot \prod_i p(d_i|c)$$

Where $p(c)$ can be estimated counting the fraction of instances in class c, and $p(d_i \mid c)$ can be fitted by maximum likelihood.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $p(d_i \mid c)$.

Bernoulli Naive Bayes

In the Bernoulli event model, features are binary variables (term is present / absent). This event model is especially popular for classifying short texts.

$$p(d_i|c) = \begin{cases} p_{i,c} & \text{if } d_i = 1 \\ 1 - p_{i,c} & \text{if } d_i = 0 \end{cases}$$

Where $p_{i,c}$ is the probability of class c generating the term i. The maximum likelihood estimator of $p_{i,c}$ is the fraction of documents containing term i in class c.

```
# class
sklearn.naive_bayes.BernoulliNB
```

Multinomial Naive Bayes

With a multinomial event model, feature vectors represent the frequencies with which certain terms have been generated by a multinomial with parameters $(p_{1,c}, ..., p_{i,c}, ...)$ where $p_{i,c}$ is the probability that term i occurs in class c.

This is the event model typically used for document classification, with events representing the occurrence of a word in a single document (although TF-IDF vectors are also known to work well in practice).

$$p(\boldsymbol{d} \mid c) = \frac{(\sum_i d_i)!}{\prod_i d_i!} \prod_i {p_{i,c}}^{d_i}$$

The maximum likelihood estimator of $p_{i,c}$ is obtained by relative frequency counting, i.e. number of times the feature appears in class c over the total count of all features for class c.

$$\hat{p}_{i,c} = \frac{N_{i,c}}{\sum_j N_{j,c}}$$

Note that a Naive Bayes classifier with a Bernoulli event model is not the same as a Multinomial Naive Bayes classifier with frequency counts truncated to one.

```
# class
sklearn.naive_bayes.MultinomialNB
```

Complement Naive Bayes

Multinomial Naive Bayes performs poorly on data sets with unbalanced classes.

Complement Naive Bayes is an adaptation of the standard Multinomial Naive Bayes algorithm that is particularly suited for imbalanced data sets.

Complement Naive Bayes regularly outperforms Multinomial Naive Bayes (often by a considerable margin) on text classification tasks.

```
# class
sklearn.naive_bayes.ComplementNB
```

Smoothing

What happens if $p(d_i|c) = 0$?

- Feature i never occurs in documents labeled c
- But then, the posterior probability $p(c|d_i)$ will be 0!

Smooth the probability estimates by adding a dummy count.

$$\hat{p}_{smooth}(d_i|c) = \frac{\alpha + N_{i,c}}{\sum_j (N_{j,c} + \alpha)}$$

The smoothing parameter α accounts for features not present in the learning samples and prevents zero probabilities in further computations.

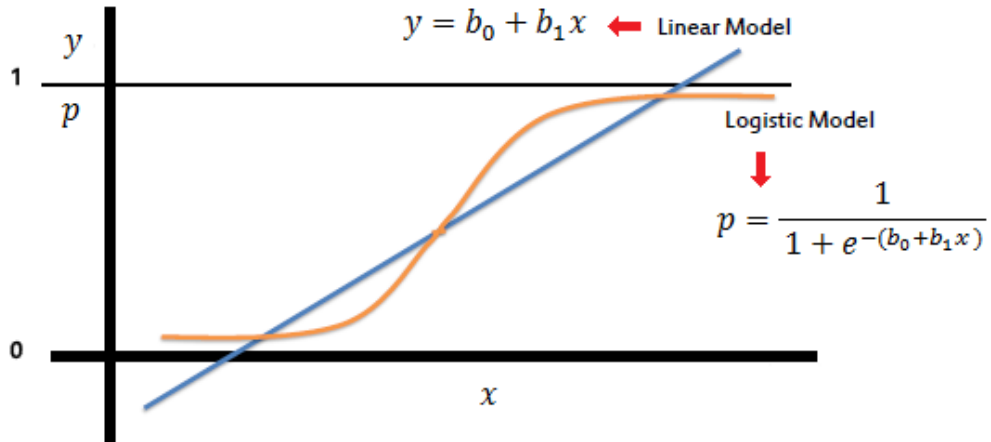
Classification Algorithms | Logistic Regression

Logistic Regression

The model itself simply models probability of output in terms of input, and does not perform statistical classification (it is not a classifier), though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other; this is a common way to make a binary classifier.

$$ln\left(\frac{p}{1-p}\right) = \beta \cdot \boldsymbol{d}$$

The regression coefficients are usually estimated using maximum likelihood estimation.


$y = b_0 + b_1 x$
Linear Model
y
1
p
Logistic Model
$p = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$
0
x

Multinomial Logistic Regression

To arrive at the multinomial logit model, one can imagine, for n possible classes, running $n-1$ independent binary logistic regression models, in which one class is chosen as a "pivot" and then the other $n-1$ classes are separately regressed against the pivot.

This would proceed as follows, if class c_n (the last one) is chosen as the pivot, then for each $0 < i < n$:

$$ln\left(\frac{p(c_i)}{p(c_n)}\right) = \boldsymbol{\beta_i} \cdot \boldsymbol{d}$$

If we exponentiate both sides, and solve for the probabilities, we get:

$$p(c_i) = p(c_n)e^{\boldsymbol{\beta_i} \cdot \boldsymbol{d}}$$

Using the fact that all n of the probabilities must sum to one, we find the probability of the last class:

$$p(c_n) = 1 - \sum_{i=1}^{n-1} p(c_i) = 1 - \sum_{i=1}^{n-1} p(c_n)e^{\boldsymbol{\beta_i} \cdot \boldsymbol{d}} \Rightarrow$$

$$\Rightarrow p(c_n) = \frac{1}{1 + \sum_{i=1}^{n-1} e^{\boldsymbol{\beta_i} \cdot \boldsymbol{d}}}$$

We can use this to find the other probabilities:

$$p(c_i) = \frac{e^{\boldsymbol{\beta_i} \cdot \boldsymbol{d}}}{1 + \sum_{j=1}^{n-1} e^{\boldsymbol{\beta_j} \cdot \boldsymbol{d}}}$$

In order to build the classifier, we might now select the label corresponding to the class with highest probability.

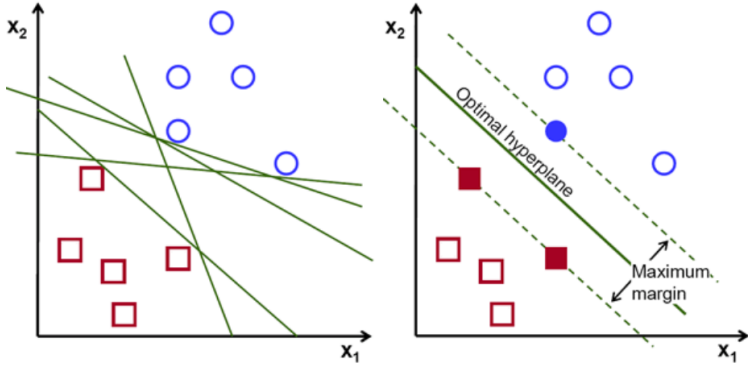
```
# class
sklearn.linear_model.LogisticRegression
```

Classification Algorithms | Support Vector Machines

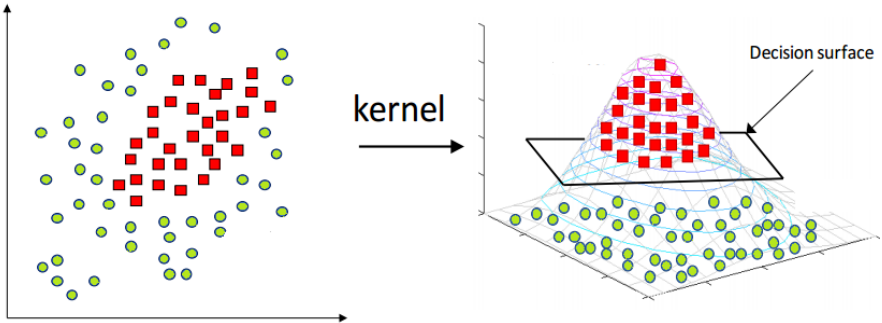
Support Vector Machines

The main principle of SVM is to determine separators in the search space which can best separate the different classes.

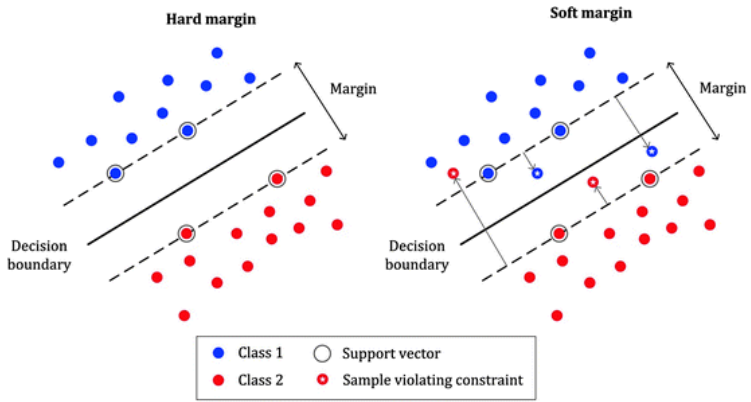
Thus SVM tries to make a decision boundary in such a way that the separation between the two classes is as wide as possible.



We note that it is not necessary to use a linear function for the SVM classifier. Rather, with the kernel trick, SVM can construct a nonlinear decision surface in the original feature space by mapping the data instances non-linearly to a new space where the classes can be separated linearly with a hyperplane. However, in practice, linear SVM is used most often because of their simplicity and ease of interpretability.



As most of real world data are not fully linearly separable, we will allow some margin violation to occur. Margin violation means choosing an hyperplane, which can allow some data points to stay in either incorrect side of hyperplane and between margin and correct side of hyperplane. This type of classification is called soft margin classification.



SVM Advantages

- SVM is quite robust to high dimensionality.
- It has been noted in that text data is ideally suited for SVM classification because of the sparse high-dimensional nature of text, in which few features are irrelevant, but they tend to be correlated with one another and generally organized into linearly separable categories.
- The SVM classifier has also been shown to be useful in large scale scenarios in which a large amount of unlabeled data and a small amount of labeled data is available.

```
# class
sklearn.svm.SVC
sklearn.svm.LinearSVC
```

Classification Algorithms | Decision Tree

Decision Tree

A decision tree is a hierarchical decomposition of the (training) data space, in which a condition on the feature value is used in order to divide the data space hierarchically.

Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items.

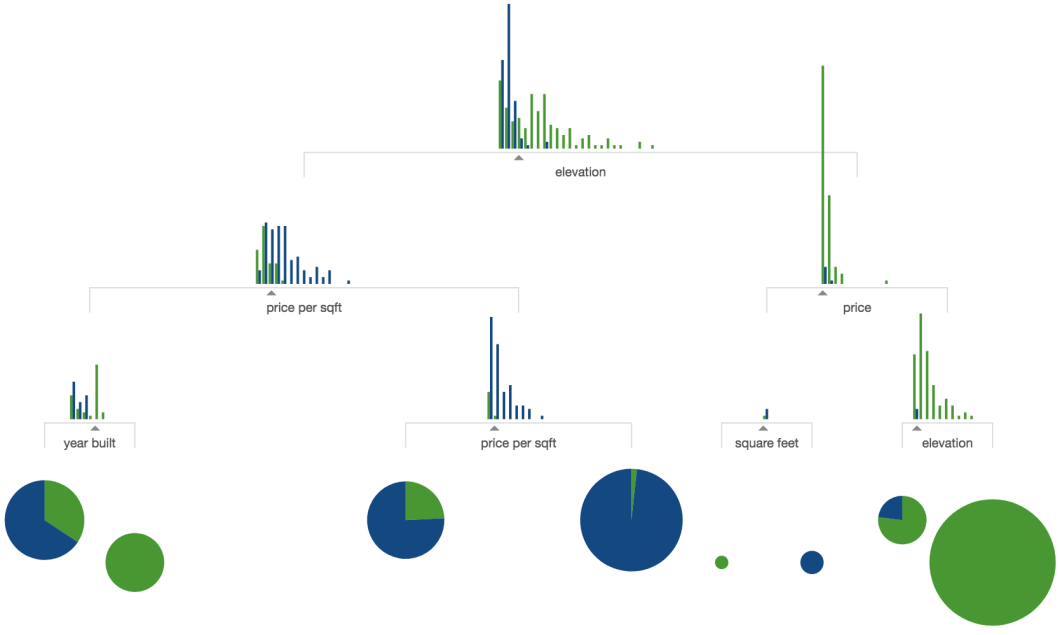
Different algorithms use different metrics for measuring “best”, such as Gini impurity or information gain. These measure the homogeneity of the target variable within the subsets.

The division of the data space is performed recursively in the decision tree, until the leaf nodes contain a certain minimum number of records, or some conditions on class purity.

For a given test instance, we apply the sequence of conditions at the nodes, in order to traverse a path of the tree in top-down fashion and determine the relevant leaf node.

The majority (weighted) class label in the leaf node is used for the purposes of classification.

```
# class
sklearn.tree.DecisionTreeClassifier
```


elevation
price per sqft
price
year built
price per sqft
square feet
elevation

Classification Algorithms | Random Forest

Random Forest

Random forests or random decision forests are an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees.

While the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the **trees are not correlated**.

Random forests correct for decision trees' habit of overfitting to their training set.

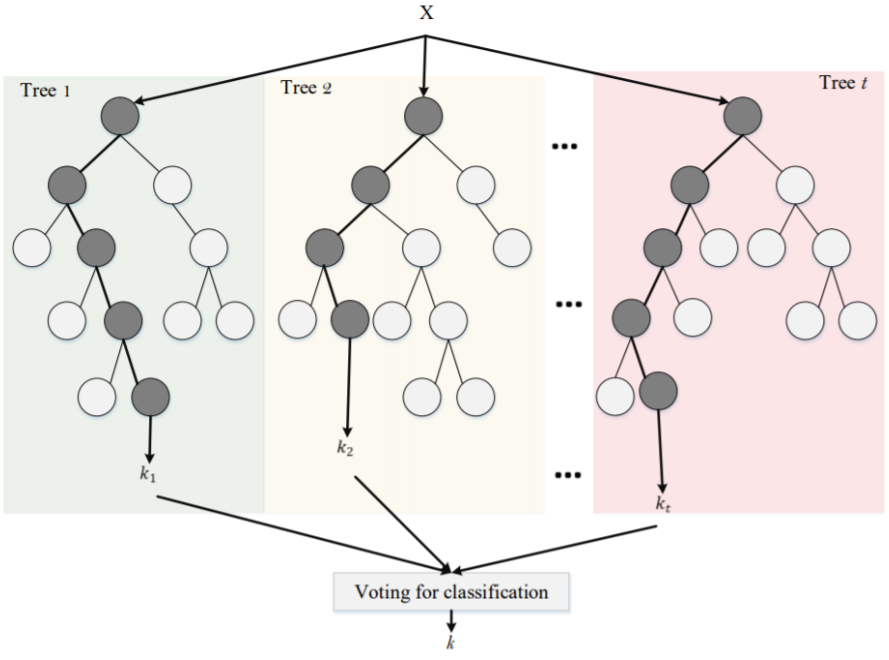
```
# class
sklearn.ensemble.RandomForestClassifier
```

De-correlating Trees

Simply training many trees on a single training set would give strongly correlated trees, or even the same tree many times. How to de-correlate trees?

- *Bagging*: given a training set, repeatedly selects a random sample with replacement of the training set and fits trees to these samples.
- *Feature Bagging*: select, at each candidate split in the learning process, a random subset of the features. Typically, for a classification problem with p features, $\sqrt{p}$ (rounded down) features are used in each split.

After training, predictions for unseen samples can be made by taking the majority vote from all the individual trees.


X
Tree 1
Tree 2
Tree t
k_1
k_2
k_t
Voting for classification
k

Classification Algorithms | Ensemble Methods

Ensemble Methods

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to enhance the accuracy of a single classification algorithm.

- Bagging
- Boosting
- Hard/Soft Voting
- Stacking

Bagging

Bagging methods form a class of algorithms which build several instances of a black-box estimator on random subsets of the original training set and then aggregate their individual predictions to form a final prediction.

The final classification is the one most often predicted by the models trained on the different random subsets of data.

As they provide a way to reduce overfitting, bagging methods work best with strong and complex models (e.g., fully developed decision trees), in contrast with boosting methods which usually work best with weak models (e.g., shallow decision trees)

```
# class
sklearn.ensemble.BaggingClassifier
```

Boosting

Boosting refers to a family of algorithms that are able to convert weak learners to strong learners.

The main principle of boosting is to fit a sequence of weak learners - models that are only slightly better than random guessing, such as shallow decision trees - to weighted versions of the data. More weight is given to examples that were misclassified by earlier rounds.

The predictions are then combined through a weighted majority vote to produce the final prediction.

```
# class
sklearn.ensemble.AdaBoostClassifier
```

Hard/Soft Voting

The idea is to combine conceptually different machine learning classifiers and use a majority vote (hard vote) or the average predicted probabilities (soft vote) to predict the class labels.

In hard voting, the predicted class label for a particular sample is the class label that represents the (weighted) majority of the class labels predicted by each individual classifier.

In contrast, soft voting returns the class label with the highest (weighted) average predicted probability.

Specific weights can be assigned to each classifier.

```
# class
sklearn.ensemble.VotingClassifier
```

Stacking

Stacking is an ensemble learning technique that combines multiple classification models via a meta-classifier.

The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features.

In other words, the meta-classifier learns how to combine the base level models to form the final prediction.

```
# class
sklearn.ensemble.StackingClassifier
```

Multiclass and Multilabel Classification

Classification Tasks

	Number of classes	Number of labels
Binary Classification	2	1
Multi-class Classification	any	1
Multi-label Classification	any	any

Multiclass classification is the task of classifying instances into one of three or more classes. Classifying instances into one of two classes is called binary classification. Multiclass classification should not be confused with multi-label classification, where multiple labels are to be predicted for each instance.

One VS Rest

While some classification algorithms naturally permit the use of more than two classes and/or labels, others are by nature binary algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies.

A common strategy is **One vs Rest**, which involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives.

```
# class
sklearn.multiclass.OneVsRestClassifier
```

In multiclass, One vs Rest requires the base classifiers to produce a real-valued score for its decision, rather than just a class label. Then, the final label is the one corresponding to the class with the highest score.

In multilabel, this strategy predicts all labels for this sample for which the respective classifiers predict a positive result.

Note that in both cases, even if the class distribution is balanced in the training set, the binary classification learners see unbalanced distributions because typically the set of negatives they see is much larger than the set of positives.

Cross Validation

Test set

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data.

This situation is called overfitting. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a **test set**.

```
# class
sklearn.model_selection.train_test_split
```

Validation Set

When evaluating different settings ("hyperparameters") for estimators, e.g. regularization parameters, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally.

This way, knowledge about the test set can "leak" into the model and evaluation metrics no longer report on generalization performance.

To solve this problem, yet another part of the dataset can be held out as a so-called **validation set**.

Training proceeds on the **training set**, after which evaluation is done on the **validation set**, and when the experiment seems to be successful, final evaluation can be done on the **test set**.

Cross Validation

By partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

A solution to this problem is a procedure called **Cross Validation**, which involves considering multiple choices for the pair (train, validation) sets and averaging evaluation metrics.

Cross validation can be used to perform model selection using Grid Search for the optimal hyperparameters of the model.

```
# class
sklearn.model_selection.GridSearchCV
```

k-fold

A test set should still be held out for final evaluation, but the validation set is no longer needed when doing cross validation.

In the basic approach, called k-fold, the training set is split into k smaller sets and:

- A model is trained using $k - 1$ of the folds as training data.
- The model is evaluated on the remaining part of the data.

The performance measure reported by k-fold is then the average of the values computed in the loop.

k-fold Variations

Repeated k-fold: repeats k-fold n times. It can be used when one requires to run k-fold n times, producing different splits in each repetition.

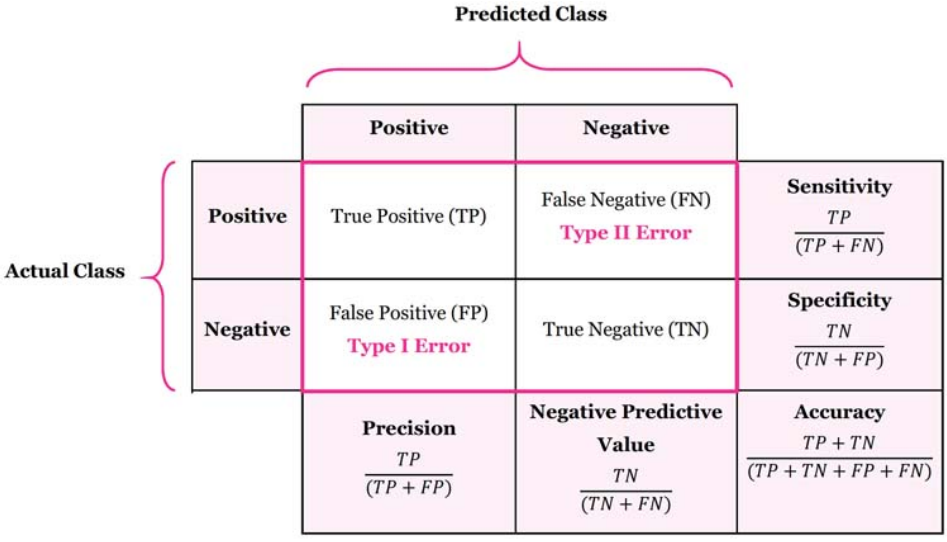
Stratified k-fold: each set contains approximately the same percentage of samples of each target class as the complete set.

Leave One Out: each training set is created by taking all the samples except one, the validation set being the sample left out. Thus, for n samples, we have n different training sets and n different validation set.

Leave p Out: it creates all the possible training/validation sets by removing p samples from the complete set.

Evaluation Metrics

Confusion Matrix



Accuracy

What proportion of instances is correctly classified?

$$\frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy is a valid choice of evaluation for classification problems which are well balanced and not skewed.

Let us say that our target class is very sparse. Do we want accuracy as a metric of our model performance? What if we are predicting if an asteroid will hit the earth? Just say "No" all the time. And you will be 99% accurate. The model can be reasonably accurate, but not at all valuable.

Precision

What proportion of predicted positives is truly positive?

$$\frac{TP}{TP + FP}$$

Precision is a valid choice of evaluation metric when we want to be very sure of our prediction.

For example: If we are building a system to predict if we should decrease the credit limit on a particular account, we want to be very sure about our prediction or it may result in customer dissatisfaction.

Recall

What proportion of truly positives is correctly classified?

$$\frac{TP}{TP + FN}$$

Recall is a valid choice of evaluation metric when we want to capture as many positives as possible.

For example: If we are building a system to predict if a person has cancer or not, we want to capture the disease even if we are not very sure.

F1 Score

The F1 score is the harmonic mean of precision and recall. Simply stated the F1 score sort of maintains a balance between the precision and recall.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

If you are a police inspector and you want to catch criminals, you want to be sure that the person you catch is a criminal (precision) and you also want to capture as many criminals (recall) as possible. The F1 score manages this tradeoff.

The main problem with the F1 score is that it gives equal weight to precision and recall. We might sometimes need to include domain knowledge in our evaluation where we want to have more recall or more precision.

To solve this, we can do this by creating a weighted F1 metric as below where β manages the tradeoff between precision and recall.

$$F_\beta = (1 + \beta^2) \cdot \frac{\textit{precision} \cdot \textit{recall}}{(\beta^2 \cdot \textit{precision}) + \textit{recall}}$$

Here we give β times as much importance to recall as precision.

AUC Score

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative').

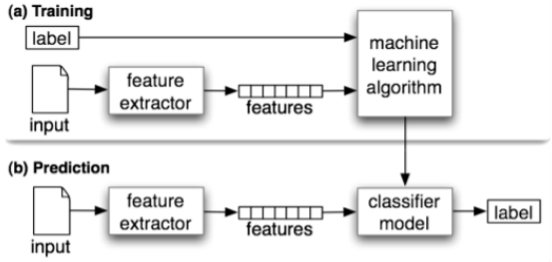
AUC measures how well predictions are ranked, rather than their absolute values. AUC ranges in value from 0 (100% wrong), to 0.5 (random classifier), to 1 (100% correct).

So, for example, if you as a marketer want to find a list of users who will respond to a marketing campaign, AUC is a good metric to use since the predictions ranked by probability is the order in which you will create a list of users to send the marketing campaign.

Take Home Concepts

Take Home Concepts

- Split the data in training and test set
- (training set) Extract and select relevant features from text
- (training set) Learn the best model via cross validation
- (test set) Extract the features
- (test set) Make predictions and evaluate the final model



References

References

- https://towardsdatascience.com/how-i-improved-my-text-classification-model-with-feature-engineering-98fbe6c13ef3
  - read (11 mins)
- https://www.nltk.org/book/ch06.html
  - 1, 2, 3, 4, 5
- https://arxiv.org/pdf/1904.08067.pdf
  - 1, 2.1, 2.2, 2.3, 3.1.1, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 6.3.1
- https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989
  - read (7 mins)
- https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226
  - read (9 mins)