Feature Selection

Classification of feature selection approaches

https://sebastianraschka.com/faq/docs/feature_sele_categories.html (https://sebastianraschka.com/faq/docs/feature_sele_categories.html)

Filter methods

The filter methods pick up the intrinsic properties of the features (i.e., the “relevance” of the features) measured via **univariate statistics** instead of cross-validation performance.

- variance threshold
- χ^2 method;
- fisher score
- correlation coefficient: Remove the correlated variables prior to selecting important variables this is different from feature extraction with, e.g. PCA.
- information gain: See the book 'An introduction to information retrieval'.

Example:

Here, we simply compute the variance of each feature, and we select the subset of features based on a user-specified threshold. E.g., “keep all features that have a variance greater or equal to x” or “keep the the top k features with the largest variance.” We assume that features with a higher variance may contain more useful information, **but note that we are not taking the relationship between feature variables or feature and target variables into account, which is one of the drawbacks of filter methods.** Comments: in chi-square test, we do consider the relationship between feature and target.

Wrapper methods:

Wrapper methods measure the “usefulness” of features based on the classifier performance. So, wrapper methods are essentially solving the “real” problem (optimizing the classifier performance), but they are also computationally more expensive compared to filter methods due to the repeated learning steps and cross-validation.

- sequential feature selection algorithms

- recursive feature elimination
- genetic algorithms

Example: Sequential Feature Selection

Sequential Forward Selection (SFS), a special case of sequential feature selection, is a greedy search algorithm that attempts to find the “optimal” feature subset by iteratively selecting features based on the classifier performance. We start with an empty feature subset and add one feature at the time in each round; this one feature is selected from the pool of all features that are not in our feature subset, and it is the feature that – when added – results in the best classifier performance. Since **we have to train and cross-validate our model for each feature subset combination**, this approach is much more expensive than a filter approach such as the variance threshold, which we discussed above.

Embedded methods:

Embedded methods, are quite similar to wrapper methods since they are also used to optimize the objective function or performance of a learning algorithm or model. The difference to wrapper methods is that an intrinsic model building metric is used during learning.

- L1 (LASSO) regularization
- Use Random Forest, Xgboost and plot variable importance chart

Other methods

- Exploratory data analysis, particularly with histogram, etc.

Removing strongly correlated features

https://chrisalbon.com/machine_learning/feature_selection/drop_highly_correlated_features/
(https://chrisalbon.com/machine_learning/feature_selection/drop_highly_correlated_features/)

In [1]:
```
import pandas as pd
import numpy as np
X = np.array([[1, 1, 1],
              [2, 2, 0],
              [3, 3, 1],
              [4, 4, 0],
              [5, 5, 1],
              [6, 6, 0],
              [7, 7, 1],
              [8, 7, 0],
              [9, 7, 1]])

df = pd.DataFrame(X)
df
```

Out[1]:

	0	1	2
0	1	1	1
1	2	2	0
2	3	3	1
3	4	4	0
4	5	5	1
5	6	6	0
6	7	7	1
7	8	7	0
8	9	7	1

In [4]:
```
corr_matrix = df.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
print(upper)

# Find index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]
```

```
     0         1         2
0  NaN  0.976103  0.000000
1  NaN       NaN  0.034503
2  NaN       NaN       NaN
```

In [3]:
```
# Drop features
df.drop(df.columns[to_drop], axis=1)
```

Out[3]:

	0	2
0	1	1
1	2	0
2	3	1
3	4	0
4	5	1
5	6	0
6	7	1
7	8	0
8	9	1

Feature selection with χ^2 test

This is for the cases involving categorical features. However, it is not that whenever categorical variables are used, then we need χ^2 test, as in the notes of "forward step-wise feature selection using ROC_AUC.pdf".

χ^2 distribution and two assumptions to do χ^2 test

Check the statistics notes in mathematics folder about how the sum of square of normally distributed and independent random variables gives χ^2 distribution, how it is related to for example γ distribution, and how χ^2 approaches Gaussian distribution as the degree of freedom becomes very large...

https://en.wikipedia.org/wiki/Chi-squared_test (https://en.wikipedia.org/wiki/Chi-squared_test)
A chi-squared test, also written as χ^2 test, is any statistical hypothesis test where the sampling distribution of the test statistic is a χ^2 distribution when the null hypothesis is true. Without other qualification, 'chi-squared test' often is used as short for Pearson's chi-squared test. The chi-squared test is used to determine whether there is a significant difference between the expected frequencies and the observed frequencies in one or more categories.

Check the link https://en.wikipedia.org/wiki/Chi-squared_test (https://en.wikipedia.org/wiki/Chi-squared_test) and https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test (https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test) about how the χ^2 statistic was constructed by Pearson and why that statistic follows the χ^2 distribution under **two important conditions**. Because we need independently normally distributed data, we need large number in each cell of contingency table (in practice > 5) for normal distribution (central limit theorem) and also require that the data is independently sampled.

Intuition for χ^2 test

Large values of χ^2 indicate that observed and expected frequencies are far apart. Small values of χ^2 mean the opposite: observed are close to expected. So χ^2 does give a measure of the distance between observed and expected. The variables are considered **independent** if the observed and expected frequencies are similar.
Comments:

- The smaller the χ^2, the smaller distance, and thus the more independent. If χ^2 is very big, then usually the p value calculated will be very tiny, and thus we can reject the null hypothesis: data are independent or observed and expected frequency are similar.
- When feature and target are independent, then discard the feature. When one feature and another feature are strongly dependent, discard one of features. This is similar when using either χ^2 or correlation efficients. However, one is suitable for numerical value and the other is suitable for categorical value.

χ^2 testing example and sample code

https://machinelearningmastery.com/chi-squared-test-for-machine-learning/ (https://machinelearningmastery.com/chi-squared-test-for-machine-learning/)

A categorical variable is a variable that may take on one of a set of labels. An example might be sex, which may be summarized as male or female. The variable is ‘sex‘ and the labels or factors of the variable are ‘male‘ and ‘female‘ in this case.

We may wish to look at a summary of a categorical variable as it pertains to another categorical variable. For example, sex and interest, where interest may have the labels ‘science‘, ‘math‘, or ‘art‘. We can collect observations from people collected with regard to these two categorical variables; for example:

```
 Sex,    Interest
Male,    Art
Female,    Math
Male,     Science
Male,    Math
```

We can summarize the collected observations in a table with one variable corresponding to columns and another variable corresponding to rows. Each cell in the table corresponds to the count or frequency of observations that correspond to the row and column categories. Historically, a table summarization of two categorical variables in this form is called a contingency table. **Try automatically implement this by pivot table in either EXCEL or Pandas.**

```
 Science,    Math,    Art
Male         20,      30,    15
Female       20,      15,    30
```

The expected frequency in cell (2,3) is calculated as (total in row 2)(total in column 3)/(total in all rows and columns) = 65x45/130 = 22.5. The contribution from cell (2,3) to the statistic χ^2 is thus (30-22.5)^2/22.5 = 2.5. The contributions from all other cells can be calculated in a similar way and thus we can obtain the χ^2.

The degree of freedom (dof) in this case is (3-1)(2-1) = 2. Then from the dof=2 χ^2 distribution curve, calculate the probability to have $\chi^2 \geq 2.5$, which is the p value. If this p value is very tiny as compared to a critical value, then we reject the null hypothesis: the two categorical variables are with similar frequency, or independent.

In the link https://stattrek.com/chi-square-test/independence.aspx (https://stattrek.com/chi-square-test/independence.aspx) or in the downloaded 'chi-square test of independence.pdf', there is a similar example but in more detail. More mathematical more details can be found in the linke http://nlp.stanford.edu/IR-book/html/htmledition/feature-selectionchi2-feature-selection-1.html (http://nlp.stanford.edu/IR-book/html/htmledition/feature-selectionchi2-feature-selection-1.html) or in the book https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf (https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf)

Finally note when we calculate the expected frequency of each cell for a single categorical variable, it is normally just the average frequency (for equal probability) but not the value calcuated the way we used earlier. See an example in the following link https://www.khanacademy.org/math/ap-statistics#tests-significance-ap (https://www.khanacademy.org/math/ap-statistics#tests-

significance-ap).

In [1]:
```
# chi-squared test with similar proportions
from scipy.stats import chi2_contingency
from scipy.stats import chi2
# contingency table
table = [   [10, 20, 30],
            [6,  9,  17]]
print(table)
stat, p, dof, expected = chi2_contingency(table)
#stat is statistic, i.e. chi2 here; p is p value; dof is degree of freedom, and expected is expected frequency.
print('dof=%d' % dof)
print(expected)
# interpret test-statistic
prob = 0.95
critical = chi2.ppf(prob, dof)
print('probability=%.3f, critical=%.3f, stat=%.3f' % (prob, critical, stat))
if abs(stat) >= critical:
    print('Dependent (reject H0)')
else:
    print('Independent (fail to reject H0)')
# interpret p-value
alpha = 1.0 - prob
print('significance=%.3f, p=%.3f' % (alpha, p))
if p <= alpha:
    print('Dependent (reject H0)')
else:
    print('Independent (fail to reject H0)')
```

```
[[10, 20, 30], [6, 9, 17]]
dof=2
[[10.43478261 18.91304348 30.65217391]
 [ 5.56521739 10.08695652 16.34782609]]
probability=0.950, critical=5.991, stat=0.272
Independent (fail to reject H0)
significance=0.050, p=0.873
Independent (fail to reject H0)
```

Comments:

- In the above example, normally we need just use p value, but not need both.

- When categorical is with "label encoding" or with "one hot encoding", we should do χ^2 test when necessary and when the numbers in the contingency table is much bigger than 5. **However, how χ^2 performs in two different cases, one is with label encoding and the other is with one-hot encoding?**

Testing the relation of between categorical and continuous variables

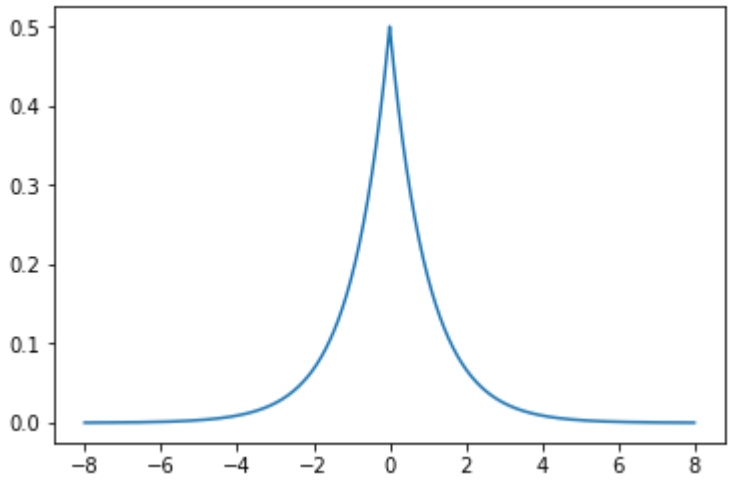
- This is often done by discretizing the continuous variable, and then still use the χ^2 test.
- We can use ANCOVA (analysis of covariance) technique to capture association between continuous and categorical variables.
- χ^2 vs ANCOVA
  - The chi square and Analysis of Variance (ANOVA) are both inferential statistical tests. Inferential statistics are used to determine if observed data we obtain from a sample (i.e., data we collect) are different from what one would expect by chance alone.
  - That said, chi square is used when we have two categorical variables (e.g., gender and alive/dead) and want to determine if one variable is related to another. In ANOVA, we have two or more group means (averages) that we want to compare. In an ANOVA, one variable must be categorical and the other must be continuous.
  - Although chi square can also handle the relation between one continuous and one categorical, how does it performs as compared to ANCOVA?

Feature selection using l_1 regularization

- l_1 (or LASSO) regression for generalized linear models can be understood as adding a penalty against complexity to reduce the degree of overfitting or variance of a model by adding more bias. Here, we add a penalty term directly to the cost function. Through adding the l_1 term, our objective function now becomes the minimization of the regularized cost, and since the penalty term grows with the value of the weight parameters (λ is just a free parameter to fine-tune the regularization strength), we can induce sparsity through this L1 vector norm, which can be considered as an intrinsic way of feature selection that is part of the model training step.
- Unlike l_2 regularization imposing a Gaussian prior to the weights, l_1 regularization restricts the weights with a Laplace prior. Due to the special feature of Laplace distribution, l_1 regularization is thus more selective?
- Is it possible for us to use l_1 regularization to select feature step by step? For example, eliminate the previous selected one/ones, and then select the next most important?
- See an example of choosing most important features with scikit-learn.

In [7]:
```
import matplotlib.pyplot as plt
import numpy as np
loc, scale = 0., 1
x = np.arange(-8., 8., .01)
pdf = np.exp(-abs(x-loc)/scale)/(2.*scale)
plt.plot(x, pdf)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x24840defbe0>]
```



Sequential Feature Selector

http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/
(http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/)

Implementation of sequential feature algorithms (SFAs) -- greedy search algorithms -- that have been developed as a suboptimal solution to the computationally **often not feasible** exhaustive search.
Sequential feature selection algorithms are a family of greedy search algorithms that are used to reduce an initial d-dimensional feature space to a k-dimensional feature subspace where k < d. The motivation behind feature selection algorithms is to automatically select a subset of features that is most relevant to the problem. The goal of feature selection is two-fold: We want to improve the computational efficiency and reduce the generalization error of the model by removing irrelevant features or noise. A wrapper approach such as sequential feature selection is especially useful if embedded feature selection -- for example, a regularization penalty like LASSO -- is not applicable (**When**).

In the above link, there is an API providing four types of sequential feature selecting approachs, all from the SequentialFeatureSelector:

Sequential Forward Selection (SFS)
Sequential Backward Selection (SBS)
Sequential Forward Floating Selection (SFFS)
Sequential Backward Floating Selection (SBFS)

The floating variants, SFFS and SBFS, can be considered as extensions to the simpler SFS and SBS algorithms. The floating algorithms have an additional exclusion or inclusion step to remove features once they were included (or excluded), so that a larger number of feature subset combinations can be sampled. It is important to emphasize that this step is conditional and only occurs if the resulting feature subset is assessed as "better" by the criterion function after removal (or addition) of a particular feature. Furthermore, I added an optional check to skip the conditional exclusion steps if the algorithm gets stuck in cycles.

See the link for details of examples of how the above four selector works.
Also see a similar forward selecting approach in the file "forward step-wise feature selection using ROC_AUC.pdf" in the same folder.

Feature selection in sci-kit learn

https://scikit-learn.org/stable/modules/feature_selection.html (https://scikit-learn.org/stable/modules/feature_selection.html)
The classes in the sklearn.feature_selection module can be used for feature selection/dimensionality reduction on sample sets, either to improve estimator's accuracy scores or to boost their performance on very high-dimensional datasets. 1.13. Feature selection
1.13.1. Removing features with low variance
1.13.2. Univariate feature selection
1.13.3. Recursive feature elimination
1.13.4. Feature selection using SelectFromModel
1.13.4.1. L1-based feature selection
1.13.4.2. Tree-based feature selection

Recursive Feature Elimination

The Recursive Feature Elimination (RFE) method is a feature selection approach. It works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

How is sequential feature selector introduced earlier different from Recursive Feature Elimination (RFE) -- e.g., as implemented in sklearn.feature_selection.RFE? RFE is computationally less complex using the feature weight coefficients (e.g., linear models) or feature importance (tree-based algorithms) to eliminate features recursively, whereas SFSs eliminate (or add) features based on a user-defined

classifier/regression performance metric.

This recipe shows the use of RFE on the Iris floweres dataset to select 3 attributes.

In [11]:
```
# Recursive Feature Elimination
from sklearn import datasets
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load the iris datasets
dataset = datasets.load_iris()
# create a base classifier used to evaluate a subset of attributes
model = LogisticRegression(solver = 'lbfgs', multi_class='auto',max_iter=1000 )
# create the RFE model and select 3 attributes
rfe = RFE(model, 3)
rfe = rfe.fit(dataset.data, dataset.target)
# summarize the selection of the attributes
print(rfe.support_)
print(rfe.ranking_)
```

```
[False  True  True  True]
[2 1 1 1]
```

Feature Importance

Methods that use ensembles of decision trees (like Random Forest or Extra Trees) can also compute the relative importance of each attribute. These importance values can be used to inform a feature selection process.

This recipe shows the construction of an Extra Trees ensemble of the iris flowers dataset and the display of the relative feature importance.

In [13]:
```
# Feature Importance
from sklearn import datasets
from sklearn import metrics
from sklearn.ensemble import ExtraTreesClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit an Extra Trees model to the data
model = ExtraTreesClassifier(n_estimators= 100)
model.fit(dataset.data, dataset.target)
# display the relative importance of each attribute
print(model.feature_importances_)
```

```
[0.10039455 0.05349868 0.40442045 0.44168633]
```

Feature extraction

Introduction

- Feature extraction can reduce dimensions and make algorithms more stable.
- Feature extraction, particular nonlinear dimension reduction, can help implement complex functions with small amount of data. If the extracted data is already a nonlinear function of the original data, then inputing these data to simple algorithm can also realize complex function.
- When large amount of data is available, we may use simple algorithm such as simple neural network to implement complex functions. When small amount of data is available, we have other options such as
  - Hand-engineer complex algorithm or network.
  - Generate nonlinear data representation by feature extraction.
  - Transfer learning.

PCA, LDA, FA

All details can be found in the notes for PCA, LDA and FA in the /algorithm folder.
Also some examples are in
http://rasbt.github.io/mlxtend/user_guide/feature_extraction/LinearDiscriminantAnalysis/

(http://rasbt.github.io/mlxtend/user_guide/feature_extraction/LinearDiscriminantAnalysis/)
http://rasbt.github.io/mlxtend/user_guide/feature_extraction/PrincipalComponentAnalysis/
(http://rasbt.github.io/mlxtend/user_guide/feature_extraction/PrincipalComponentAnalysis/)

Nonlinear dimension reduction

Feature extraction algorithms are mostly dimension reduction techniques. The PCA, LDA and FA mentioned above are examples of linear dimensional reduction (LDR). There are also many nonlinear dimension reduction algorithms (NLDR) that are very useful. For example, the neural network based NLDR such as word embedding techniques in natural language processing, image encoding algorithm (e.g. face encoding in face recognition) have proved to be very powerful. The t-distributed neighbor embedding (t-SNE) is another NLDR example which is for visualization. See more examples of NLDR in the 'nonlinear dimension reduction.pdf' under the /algorithm folder.

Comments
Feature selection and feature extraction are in fact both dimension reduction techniques.