**April 7, 2025**

**Python programming for AI lab**

**Lab practice 2**

**Feature Selection and Feature Engineering**

Feature selection – also known as variable selection, attribute selection, or variable subset selection – is a method used to select a subset of features (variables, dimensions) from an initial dataset. Feature selection is a key step in the process of building machine learning models and can have a huge impact on the performance of a model. Using correct and relevant features as the input to your model can also reduce the chance of overfitting, because having more relevant features reduces the opportunity of a model to use noisy features that don't add signal as input. Lastly, having less input features decreases the amount of time that it will take to train a model. Learning which features to select is a skill developed by data scientists that usually only comes from months and years of experience and can be more of an art than a science. Feature selection is important because it can:

* Shorten training times
* Simplify models and make them easier to interpret
* Enhances testing set performance by reducing overfitting

One important reason to drop features is the high correlation and redundancy between input variables or the irrelevancy of certain features. These input variables can thus be removed without incurring much loss of information. Redundant and irrelevant are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

Feature engineering in some ways is the opposite of feature selection. With feature selection, you remove variables. In feature engineering, you create new variables to enhance the model. In many cases, you are using domain knowledge for the enhancement.

Feature selection and feature engineering is an important component of your machine learning pipeline, and that's why a whole chapter is devoted to this topic.

By the end of this chapter, you will know:

* How to decide if a feature should be dropped from a dataset
* Learn about the concepts of collinearity, correlation, and causation
* Understand the concept of feature engineering and how it differs from feature selection
* Learn about the difference between manual feature engineering and automated feature engineering. When is it appropriate to use each one?

**Feature selection**

We will explore the components of a machine learning pipeline In the coming chapter. A critical component of the pipeline is deciding which features will be used as inputs to the model. For many models, a small subset of the input variables provide the lion's share of the predictive ability. In most datasets, it is common for a few features to be responsible for the majority of the information signal and the rest of the features are just mostly noise.

It is important to lower the amount of input features for a variety of reasons including:

• Reducing the multi collinearity of the input features will make the machine learning model parameters easier to interpret. Multicollinearity (also collinearity) is a phenomenon observed with features in a dataset where one predictor feature in a regression model can be linearly predicted from the other's features with a substantial degree of accuracy.

• Reducing the time required to run the model and the amount of storage space the model needs will allow us to run more variations of the models leading to quicker and better results.

• The smaller number of input features a model requires, the easier it is to explain it. When the number of features goes up, the explainability of the model goes down. Reducing the amount of input features also makes it easier to visualize the data when reduced to low dimensions (for example, 2D or 3D).

• As the number of dimensions increases, the possible configurations increase exponentially, and the number of configurations covered by an observation decreases. As you have more features to describe your target, you might be able to describe the data more precisely, but your model will not generalize with new data points – your model will overfit the data. This is known as the curse of dimensionality.

Let's think about this intuitively by going through an example. There is a real estate site in the US that allows real estate agents and homeowners to list homes for rent or for sale. Zillow is famous, among other things, for its Zestimate. The Zestimate is an estimated price using machine learning. It is the price that Zillow estimates a home will sell for if it was put on the market today. The Zestimates are constantly updated and recalculated. How does Zillow come up with this number? If you want to learn more about it, there was a competition on Kaggle that has great resources on the Zestimate. You can find out more here:

https://www.kaggle.com/c/zillow-prize-1

The exact details of the Zestimate algorithm are proprietary, but we can make some assumptions. We will now start to explore how we can come up with our own Zestimate. Let's come up with a list of potential input variables for our machine learning model and the reasons why they might be valuable:

* Square footage: Intuitively, the bigger the home, the more expensive it will be.
* Number of bedrooms: More rooms, more cost.
* Number of bathrooms: Bedrooms need bathrooms.
* Mortgage interest rates: If rates are low, that makes mortgage payments lower, which means potential homeowners can afford a more expensive home.
* Year built: In general, newer homes are typically more expensive than older homes. Older homes normally need more repairs.
* Property taxes: If property taxes are high, that will increase the monthly payments and homeowners will only be able to afford a less expensive home.
* House color: At first glance, this might not seem like a relevant variable, but what if the home is painted lime green?
* Zip code: Location, location, location. In real estate, where the home is located is an important determinant of price. In some cases, a house in one block can be hundreds of thousands of dollars more than a house on the next block. Location can be that important.
* Comparable sales: One of the metrics that is commonly used by appraisers and real estate agents to value a home is to look for similar properties to the "subject" property that have been recently sold or at least are listed for sale, to see what the sale price was or what the listing price currently is.
* Tax assessment: Property taxes are calculated based on what the county currently thinks the property is worth. This is publicly accessible information.

These could all potentially be variables that have high predictive power, but intuitively we can probably assume that square footage, the number of bedrooms, and number of bathrooms are highly correlated. Also, intuitively, square footage provides more precision than the number of bedrooms or the number of bathrooms.

So, we can probably drop the number of bedrooms and the number bathrooms and keep the square footage and don't lose much accuracy. Indeed, we could potentially increase the accuracy, by reducing the noise.

Furthermore, we can most likely drop the house color without losing precision. Features that can be dropped without impacting the model's precision significantly fall into two categories:

* Redundant: This is a feature that is highly correlated to other input features and therefore does not add much new information to the signal.
* Irrelevant: This is a feature that has a low correlation with the target feature and for that reason provides more noise than signal.

One way to find out if our assumptions are correct is to train our model with and without our assumptions and see what produces the better results. We could use this method with every single feature, but in cases where we have a high number of features the possible number of combinations can escalate quickly.

As we mentioned previously, exploratory data analysis can be a good way to get an intuitive understanding and to obtain insights into the dataset we are working with.

Let's analyze three approaches that are commonly used to obtain these insights.

They are:

* Feature importance
* Univariate selection
* Correlation matrix with heatmap

**Feature importance**

The importance of each feature of a dataset can be established by using this method. Feature importance provides a score for each feature in a dataset. A higher score means the feature has more importance or relevancy in relation to the output feature.

Feature importance is normally an inbuilt class that comes with Tree-Based Classifiers. In the following example, we use the Extra Tree Classifier to determine the top five features in a dataset:

**import pandas as pd**

**from sklearn.ensemble import ExtraTreesClassifier**

**import numpy as np**

**import matplotlib.pyplot as plt**

**data = pd.read\_csv("train.csv")**

**X = data.iloc[:,0:20] #independent columns**

**y = data.iloc[:,-1] # pick last column for the target feature**

**model = ExtraTreesClassifier()**

**model.fit(X,y)**

**print(model.feature\_importances\_) #use inbuilt class**

**#feature\_importances of tree based classifiers**

**#plot graph of feature importances for better visualization**

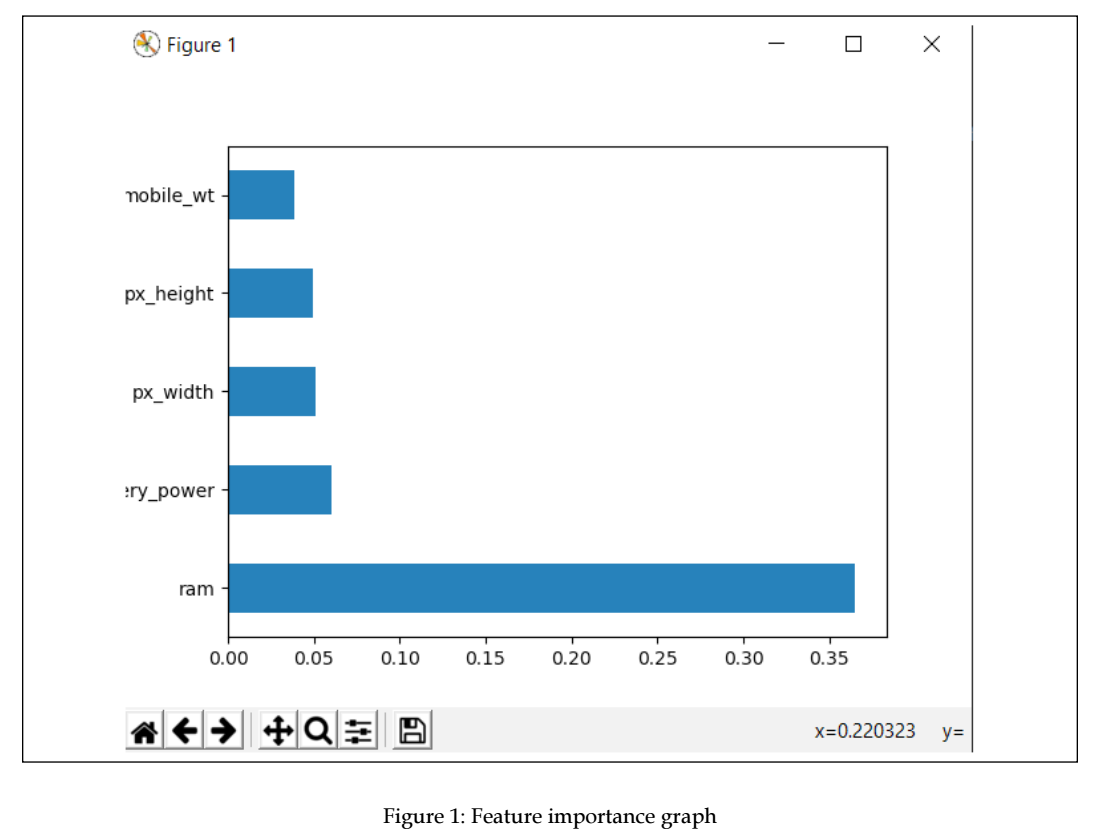
**feat\_importances = pd.Series(model.feature\_importances\_, index=X.**

**columns)**

**feat\_importances.nlargest(5).plot(kind='barh')**

**plt.show()**

You should see this as output:



**Univariate selection**

Statistical tests can be used to determine which features have the strongest correlation to the output variable. The scikit-learn library has a class called SelectKBest that provides a set of statistical tests to select the K "best" features in a dataset.

The following is an example that uses the chi-squared (chi²) statistical test for non-negative features to select the five best features in an input dataset:

**import pandas as pd**

**import numpy as np**

**from sklearn.feature\_selection import SelectKBest**

**from sklearn.feature\_selection import chi2**

**data = pd.read\_csv("train.csv")**

**X = data.iloc[:,0:20] #independent columns**

**y = data.iloc[:,-1] #pick last column for the target feature**

**#apply SelectKBest class to extract top 5 best features**

**bestfeatures = SelectKBest(score\_func=chi2, k=5)**

**fit = bestfeatures.fit(X,y)**

**dfscores = pd.DataFrame(fit.scores\_)**

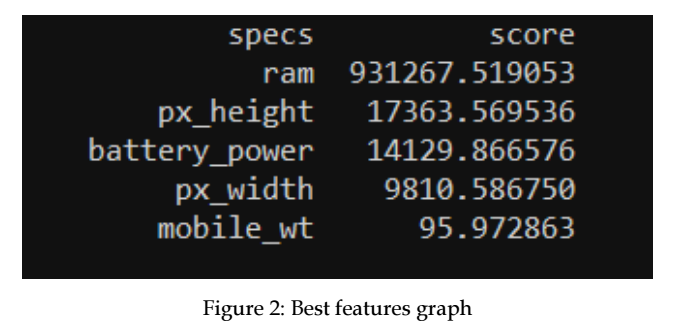
**dfcolumns = pd.DataFrame(X.columns)**

**scores = pd.concat([dfcolumns,dfscores],axis=1)**

**scores.columns = ['specs','score']**

**print(scores.nlargest(5,'score')) #print the 5 best features**

And you should see something like this as your output:



**Correlation heatmaps**

A correlation exists between two features when there is a relationship between the different values of the features. For example, if home prices go up as the square footage goes up, these two features are said to be positively correlated. There can be different degrees of correlation. If a feature changes consistently in relation to another feature, these features are said to be highly correlated.

Correlation can be positive (an increase in one value of a feature increases the value of the target variable) or negative (an increase in one value of a feature decreases the value of the target variable).

Correlation is a continuous value between -1 and 1:

* If the correlation between two variables is 1, there is a perfect direct correlation.
* If the correlation between two features is -1, a perfect inverse correlation exists.
* If the correlation is 0 between two features, there is no correlation between the two features.

A heatmap makes it easy to identify which features are most correlated to the target variable. We will plot a heatmap of correlated features using the seaborn library, using the following code:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

data = pd.read\_csv("train.csv")

X = data.iloc[:,0:20] #independent columns

y = data.iloc[:,-1] # pick last column for the target feature

#get the correlations of each feature in the dataset

correlation\_matrix = data.corr()

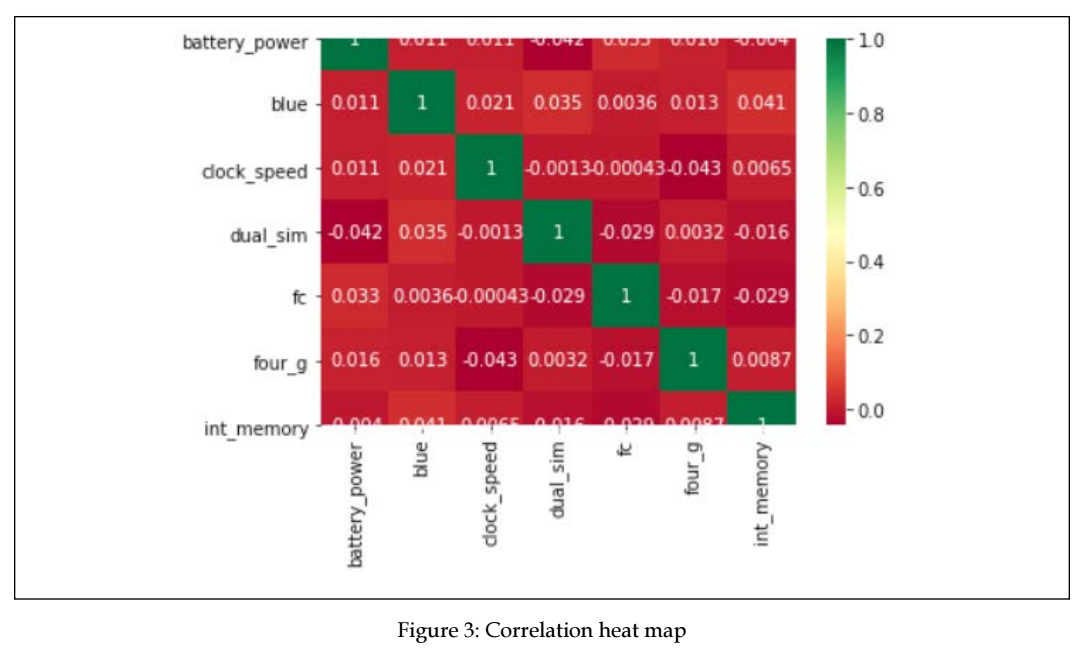
top\_corr\_features = correlation\_matrix.index

plt.figure(figsize=(20,20))

#plot heat map

g=sns.heatmap(data[top\_corr\_features].corr(),annot=True,cmap="RdYlGn")

You should get a similar output to:



More formal and less intuitive methods exist to automatically select features. Many of these methods exist and quite a few are implemented in the scikit-learn package.