EFFECTIVE DIMENSIONALITY REDUCTION TECHNIQUES

Feature Selection Techniques to Make Better Generalization Models

How to improve learning performance and increase computational efficiency

Sarit Maitra
Dec 13, 2020 · 5 min read ★

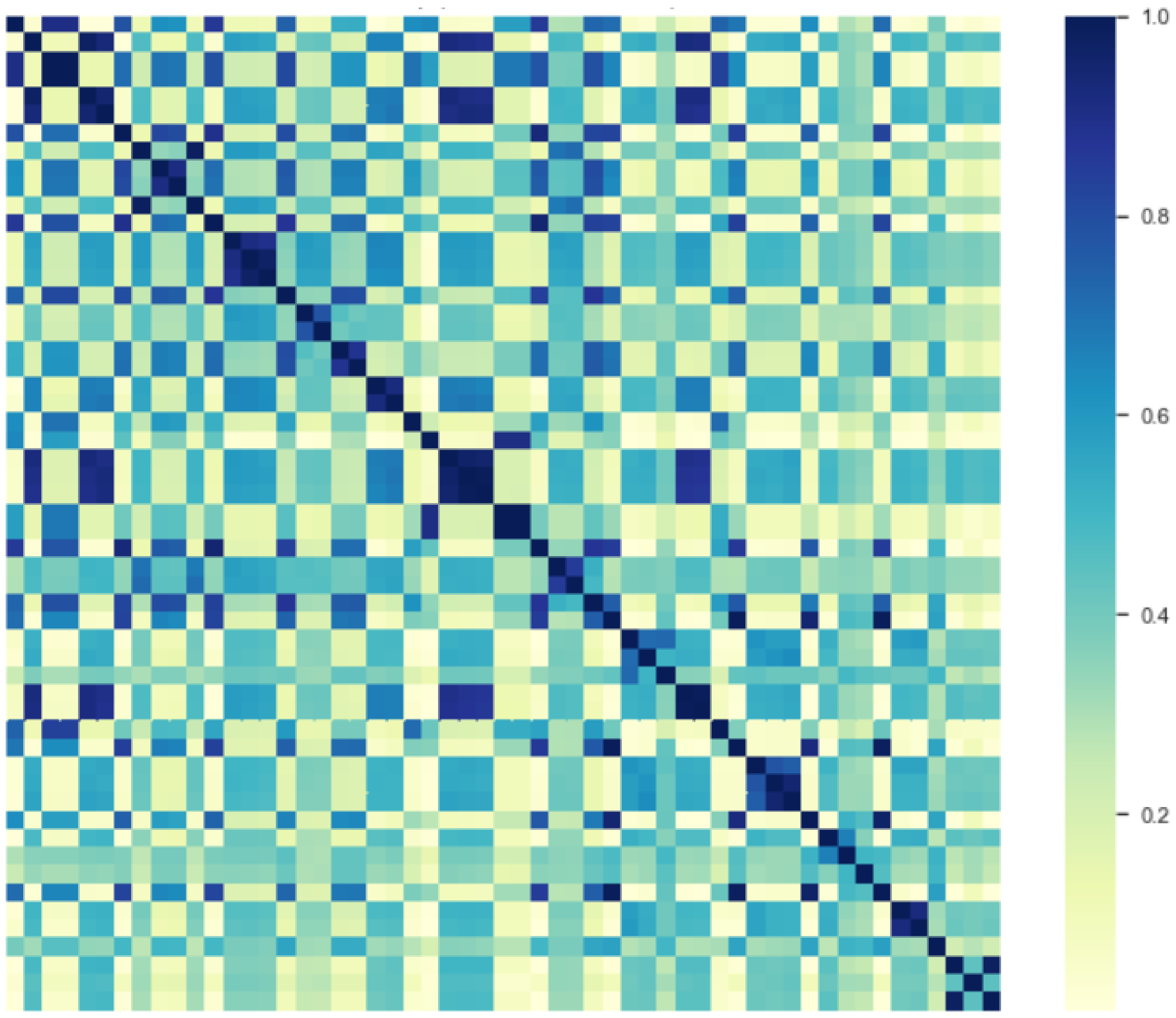


Image by author

Feature selection method is a data pre-processing step in conjunction with machine learning for classification or regression purposes. The main motivation for reducing the dimensionality of the data and keeping the number of features as low as possible is to reduce the training time and enhance the classification accuracy of the algorithms we use; moreover, reduced dimensions provide a more robust generalization and a faster response with unseen data. Unlike feature extraction, feature selection does not alter the data.

There are three main groups of feature selection in general: (1) wrapper, (2) embedded and (3) filter methods. Each group has it's own pros and cons. We will not get into the details of these methods; here we will show how different techniques including mutual information (MI is filter method) can be applied to reduce the dimensionality and still retain 99% variance in the data. Here, advantage is that, with the reduced features, noise in the dataset can be eliminated; model can easily identify the signal from the reduced and relevant dataset and learn from it.

We have a dataset with 95 features. We scale the data to standardize the features.

```
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.20,
shuffle=False)

#Summary
print("Train/Test Split Results:")
print("X Train Set:"); print(xTrain.shape)
print("X Test Set:");print(xTest.shape)

#Initialize and fit scaler
scaler = StandardScaler()
#Fit scaler using the training data
scaler.fit(xTrain)

#Transform the raw data
xTrain_st = scaler.transform(xTrain)
xTest_st = scaler.transform(xTest)
```

We will apply MI followed by Spearman's correlation and finally Principal Component Analysis to find the optimal number of features that can be used for our use case.

Mutual information (MI):

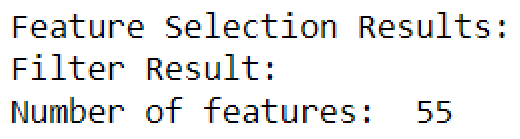
It is a measure of the amount of information that one random variable has about another variable. It can measure any kind of relation between random variables, including nonlinear relationships . MI does not make an assumption of linearity between the variables, and can deal with categorical and numerical data with two or more class values

The mutual information between two random variables x and y can be stated as: I(x ; y) = H(x) — H(x | y)

- I(x ; y) is the mutual information for x and y,
- H(x) is the entropy for x
- H(x | y) is the conditional entropy for x given y.

Entropy (H) is a measure of uncertainty of a random variable. The uncertainty is related to the probability of occurrence of an event. Intuitively, high entropy means that each event has about the same probability of occurrence, while low entropy means that each event has a different probability of occurrence. It is a non-parametric measure, therefore more the samples better for accurate estimation.

```
# FEATURE SELECTION
mi_select = SelectPercentile(mutual_info_classif,
                             percentile=60).fit(xTrain_st,
np.ravel(y_tr))
xTrain_mi = mi_select.transform(xTrain_st)
xTest_mi = mi_select.transform(xTest_st)
print("Feature Selection Results:")
print("Filter Result:"); print("Number of features: ",
xTrain_mi.shape[1])
```



```
Feature Selection Results:
Filter Result:
Number of features:  55
```

Here, we can see that, features are reduced from 95 to 55 numbers. We have used percentile as 60, which means, we have removed lower 40% of the features using MI.

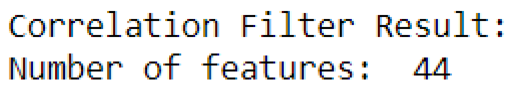
Spearman's correlation:

It is rank-order correlation is the nonparametric version of the Pearson correlation. Spearman's correlation coefficient measures the strength and direction of association between two ranked variables; it looks for monotonic relationships.

Let us apply Spearman's correlation on MI reduced dataset.

```
def correlation(xTrain, xTest, corr_threshold):
    corr = xTrain.corr(method = "spearman").abs()
    upper = corr.where(np.triu(np.ones(corr.shape), k = 
1).astype(np.bool))
    to_drop = [column for column in upper.columns if 
any(upper[column] > corr_threshold)]
    xTrain_corr_filtered = xTrain.drop(to_drop, axis = 1)
    xTest_corr_filtered = xTest.drop(to_drop, axis = 1)
    return xTrain_corr_filtered, xTest_corr_filtered

tr_mi = DataFrame(xTrain_mi,
                     index = xTrain.index,
                     columns = 
xTrain.columns[mi_select.get_support()])
te_mi = DataFrame(xTest_mi,
                    index = xTest.index,
                    columns = xTest.columns[mi_select.get_support()])
xTrain_corr, xTest_corr = correlation(tr_mi, te_mi, 0.95)
corr_before = tr_mi.corr(method = "spearman").abs()
corr_after = xTrain_corr.corr(method = "spearman").abs()
print("Correlation Filter Result:")
print("Number of features: ", xTrain_corr.shape[1])
```



```
Correlation Filter Result:
Number of features:  44
```

Here, we have reduced the features from 55 to 44 numbers.

Principal Component Analysis:

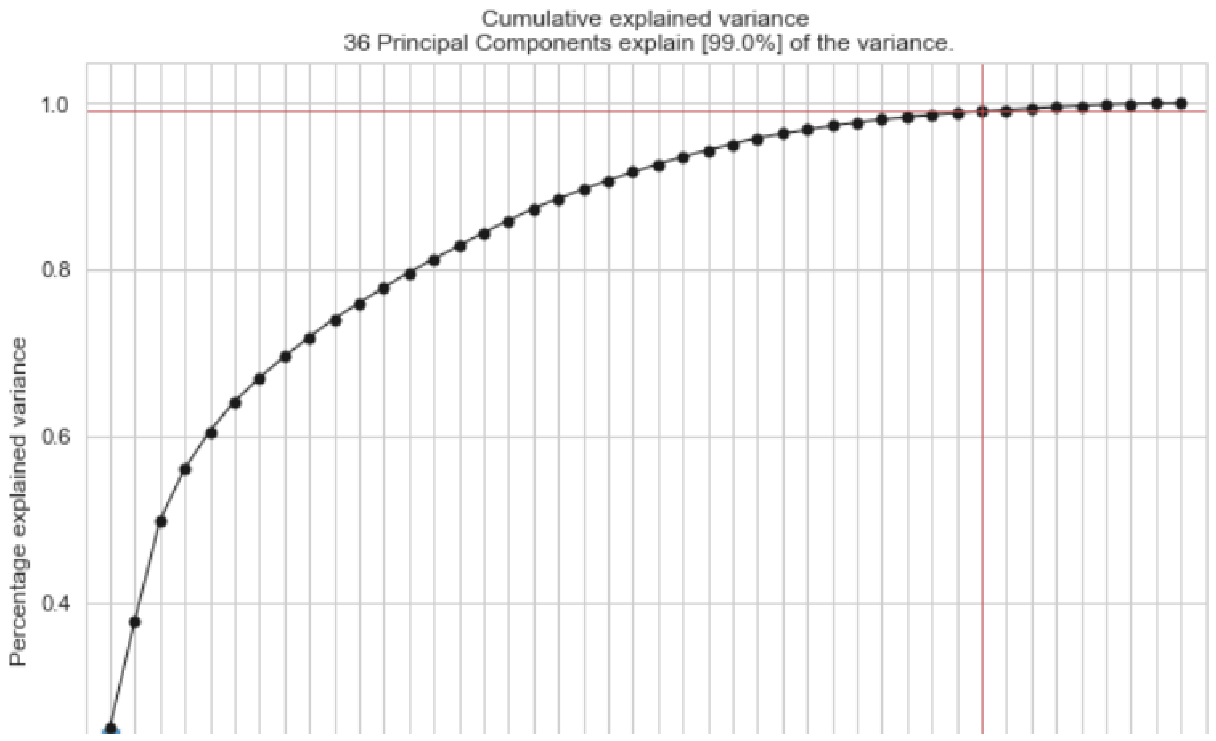
Finally we will apply principal component analysis (PCA) on the dataset obtained from Spearman's correlation. Here, we will find out optimal number of components to retain 99% variance in the data. Mathematically, PCA depends upon the eigen-decomposition of positive semi-definite matrices and upon the singular value decomposition (svd) of rectangular matrices.

```
plt.style.use('seaborn-whitegrid')
tot_var = 0.99 # total variance
model = pca(n_components = tot_var)
xTrain_pca = model.fit_transform(xTrain_corr)
# Plot explained variance
fig, ax = model.plot()

xTrain_pca = PCA(tot_var, svd_solver = 'full').fit(xTrain_corr)
print(xTrain_pca.explained_variance_ratio_); print()
print("Reduced dimensions can explain
{:.4f}".format(sum(xTrain_pca.explained_variance_ratio_)),
      "% of the variance in the original data"); print()
print(xTrain_pca.components_.shape[0]); print()
# components
n_pcs= xTrain_pca.components_.shape[0]

"""PCA coverts the features in array format; so, if we want to get
the feature names, we can use the below"""

# most important feature on each component
most_important = [np.abs(xTrain_pca.components_[i]).argmax() for i in
range(n_pcs)]
initial_feature_names = xTrain_corr.columns
most_important_names = [initial_feature_names[most_important[i]] for
i in range(n_pcs)]
dic = {'PC{}'.format(i): most_important_names[i] for i in
range(n_pcs)}
p= DataFrame(dic.items())
print(p); print()
```



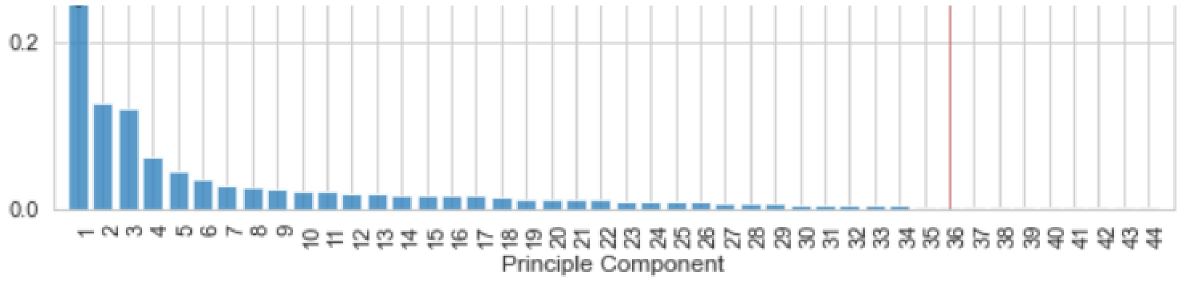


Image by author

```
[0.25098753 0.12746924 0.12021688 0.06247151 0.04478031 0.03581467
 0.02860859 0.02512014 0.02385928 0.02153757 0.01943115 0.01835669
 0.01804504 0.01618242 0.01584045 0.01571905 0.01485802 0.01434577
 0.01165293 0.01127296 0.01083339 0.01016391 0.00926412 0.00867362
 0.00821472 0.00748408 0.00720916 0.00506276 0.00491787 0.00467671
 0.00421519 0.00340941 0.00278644 0.00246482 0.00230155 0.00217166]

Reduced dimensions can explain 0.9904 % of the variance in the original data

36
```

So, we finally have reduced the number of relevant features from 95 to 36 as shown above. Now we can fit these reduced features to examine model accuracy.

Key Takeaways:

Feature selection, as a data preprocessing strategy, has been proven to be effective and efficient in preparing data (especially high-dimensional data). When we face with high dimensionality it throws us challenge in high memory consumption and computational cost in training; with a large number of features, learning models tend to overfit, which may cause performance degradation on unseen data. There are many state of the art feature selection algorithms are available; many ways to reduce the dimensionality and it all depends on the use-case and data set we will be handling. Of these, multi-variate methods generally tend to obtain better results than uni-variate approaches, but at a greater computational cost. There is no one-size-fits all method, as each is more suitable for particular kinds of problems. Intrinsic complexity of the data may influence the feature selection method. Feature selection have the advantages of improving learning performance, increasing computational efficiency, decreasing memory storage, and building better generalization models.

I can be reached *here.*

References:

1. *Vergara, J. R., & Estévez, P. A. (2014). A review of feature selection methods based on mutual information. Neural computing and applications, 24(1), 175–186.*
2. *Bennasar, M., Hicks, Y., & Setchi, R. (2015). Feature selection using joint mutual information maximisation. Expert Systems with Applications, 42(22), 8520–8532.*
3. *Li, J., & Liu, H. (2017). Challenges of feature selection for big data analytics. IEEE Intelligent Systems, 32(2), 9–15.*

Feature Selection Dimensionality Reduction Machine Learning Data Preprocessing