Feature Selection Techniques

March 7, 2022

1 Feature Selection Techniques

1.1 Introduction to Feature Selection

In machine learning and modeling, it's very common to split a single feature into several features, such hash encoding, one-hot encoding, embedding, and so on. Even though we are suppose to extract more information from the features by splitting them, there are concerns of a rapidly increasing dimension on features. For example, suppose we have 100 categorical features and each of them contains 200 classes. If we perform one-hot encoding to this dataset, we are expected to have 20,000 features, which seems totally insane for any human data scientist.

Instead of trapping into this problem of dimension, several techniques were developed for dimension requction. One of the most important goal is that before we delete some useless dimensions, we must first know the rank of the importance of the features. In this quick-note, we are going to look through several approcaches that is commonly used for feature selection and dimension reduction.

Before we begin, let's first import the functions we are going to use.

[1]:
```python
from featimp import *
import warnings
warnings.filterwarnings('ignore')
```

1.2 Common Feature Selection Algorithms

1.2.1 Spearman Rank Correlation

Spearman rank correlation is the most common algorithm for calculating the dependency of two arrays. We can use this algorithm for computing the correlation between each x_k and y and then rank the importance of the x_ks on y.

Suppose we have two arrays a and b of the same length, then the spearman rank correlation is defined by,

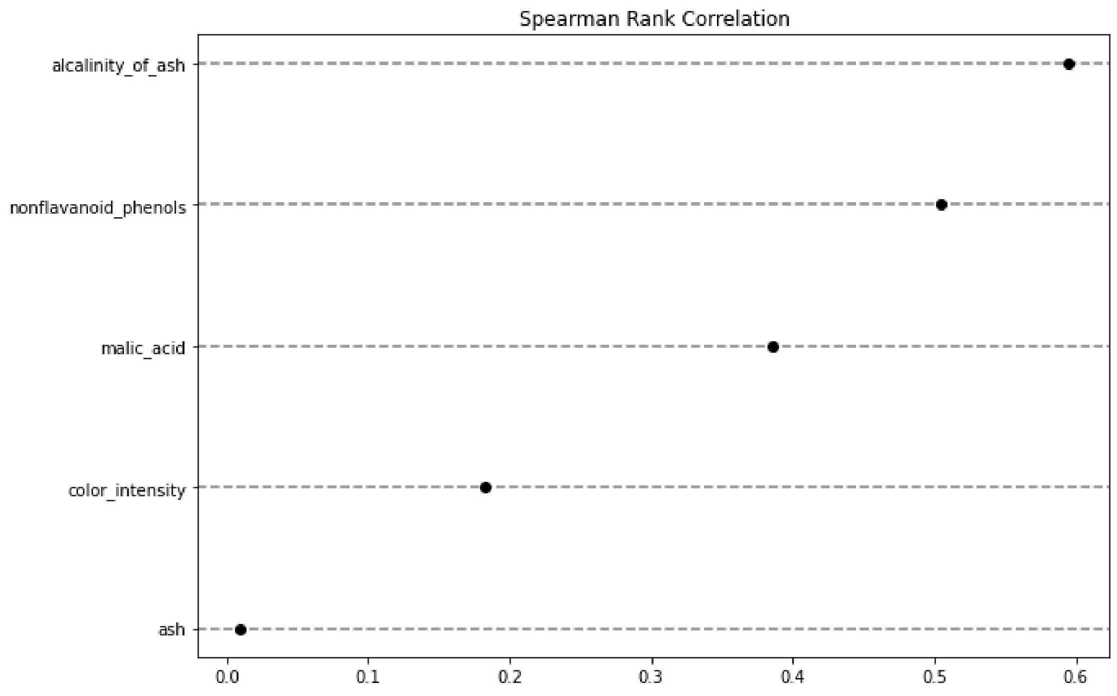
$$\rho = 1 - \frac{6\sum d_i^2}{n\left(n^2-1\right)}$$

Where d_i here is the element-wise substraction between a and b.

$$d_i = r_a^{(i)} - r_b^{(i)}$$

However, a and b are not just ordinary arrays. They should actually be the ranking array of them self. We can call the function `scipy.stats.rankdata` for getting the elements replaced by ranks of an array. To test our result, we used the toy dataset `sklearn.datasets.load_wine` for feature selection and plots. And the `plot_rank_top5` will plot the top 5 important features with the highest correlations.

```
[2]: rank = test_case_spearman()
     plot_rank_top5(rank, "Spearman Rank Correlation")
```



1.2.2 Principal component analysis (PCA)

PCA is a commonly seen algorigthm for dimension reduction. The goal of PCA is to find the principal component of a matrix. Because we have ordered the singular value so that,

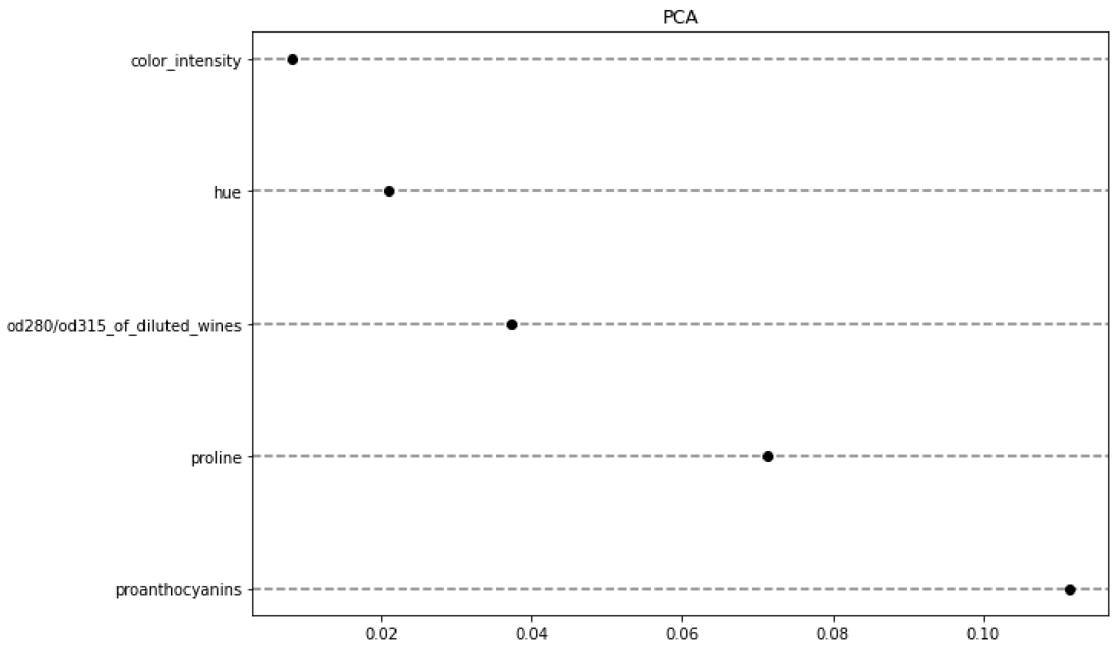
$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$$

then the express in the front is more significant for the matrix X compared with later expressions (this is just a easy way to think about and definitely not a rigorous proof). Then we can reserve the first p dimension to get an approximation of matrix X.

$$X \approx \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_p \vec{u}_p \vec{v}_p^T$$

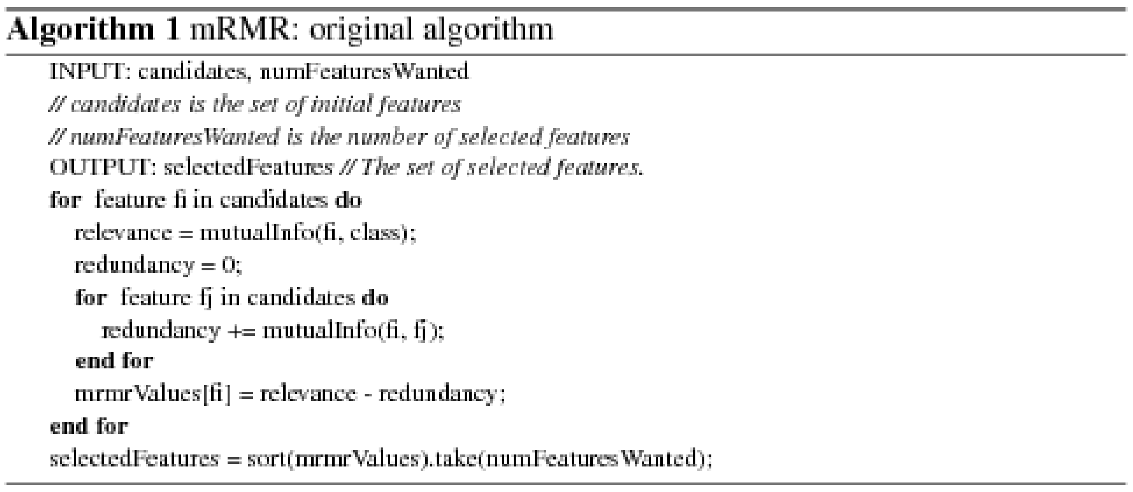
By this mean, we can reduce a r-dimension matrix to p dimensions. This algorithm works well when we have small or medium datasets. For a large dataset, this is not the best way to conduct the PCA of a matrix. The pseudo code for the PCA algorithm is, For feature selection, we don't have to get the final projected matrix because then the columns will have no practical meanings. Instead, we use the eigenvalues of the covariance matrix C to show how important a feeature is. The eigenvalues explain how much the features cover of the vatiance, which can indicate the importance of the features in the original X space.

[3]:
```
rank = test_case_pca()
plot_rank_top5(rank, "PCA")
```



1.2.3 Maximum Relevance - Minimum Redundancy (mRMR)

mRMR is another algorithm that simply considers the redundancy and dependency of a feature. The index of mRMR is created by the revalance of the feature x_k on y reduces the summerization of redundancy the feature x_k on all the other x_js. The goal is to find a feature that has the maximized dependency on y and the minimized redundancy on the other x_js.



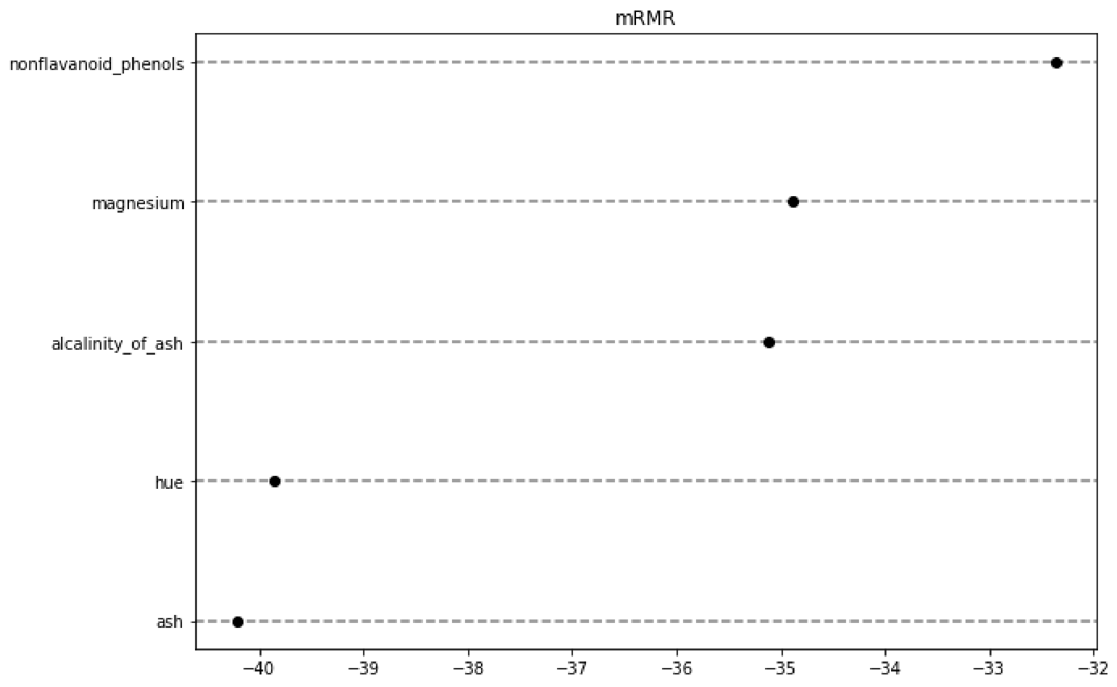
Algorithm 1 mRMR: original algorithm

```
INPUT: candidates, numFeaturesWanted
// candidates is the set of initial features
// numFeaturesWanted is the number of selected features
OUTPUT: selectedFeatures // The set of selected features.
for feature fi in candidates do
    relevance = mutualInfo(fi, class);
    redundancy = 0;
    for feature fj in candidates do
        redundancy += mutualInfo(fi, fj);
    end for
    mrmrValues[fi] = relevance - redundancy;
end for
selectedFeatures = sort(mrmrValues).take(numFeaturesWanted);
```

In this case, we have to calculate the depedencies and requndancies based on mutual information I, which can be called through the `sklearn.feature_selection.mutual_info_classif` package. We should keep in mind that this package can only be used for categorical features so that if we have continuous features, we can to use spearman correlcation to replace I.

$$J_{mRMR}\left(x_k\right) = I\left(x_k, y\right) - \frac{1}{|S|} \sum_{x_j \in S} I\left(x_k, x_j\right)$$

```
[5]: rank = test_case_mRMR()
     plot_rank_top5(rank, "mRMR")
```

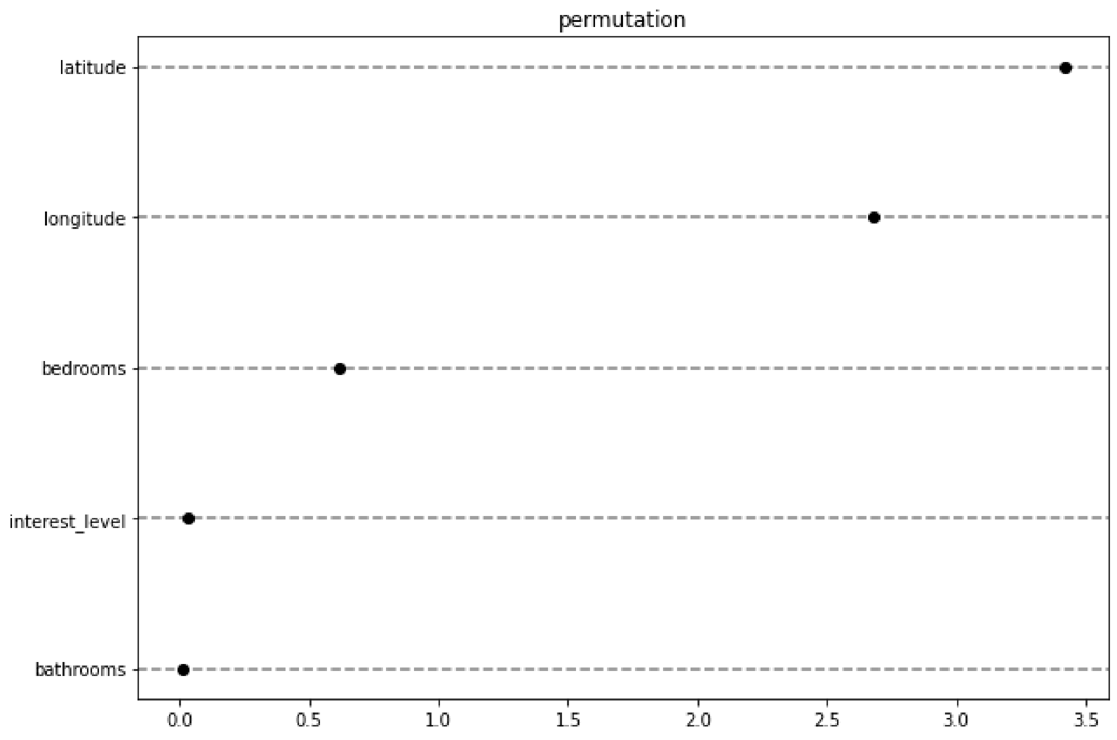


1.2.4 Permutation and Drop Column

Permutation and drop column are two model-based techniques that used for feature selection. The ideas of them are quite similar. For drop column feature importsance, we first drop the column we want to know x_k from the dataset, and then retrain the model based on the new dataset. Because we can discover a reduction of matrics we select, we are then able to know how important that feature is. For permutation, the idea is quite similar, but we don't actually remove the feature from our dataset. Instread, we take the permutation of that feature and then re-train the model. If we can observe a large change in the matrics, we can say that feature is important.

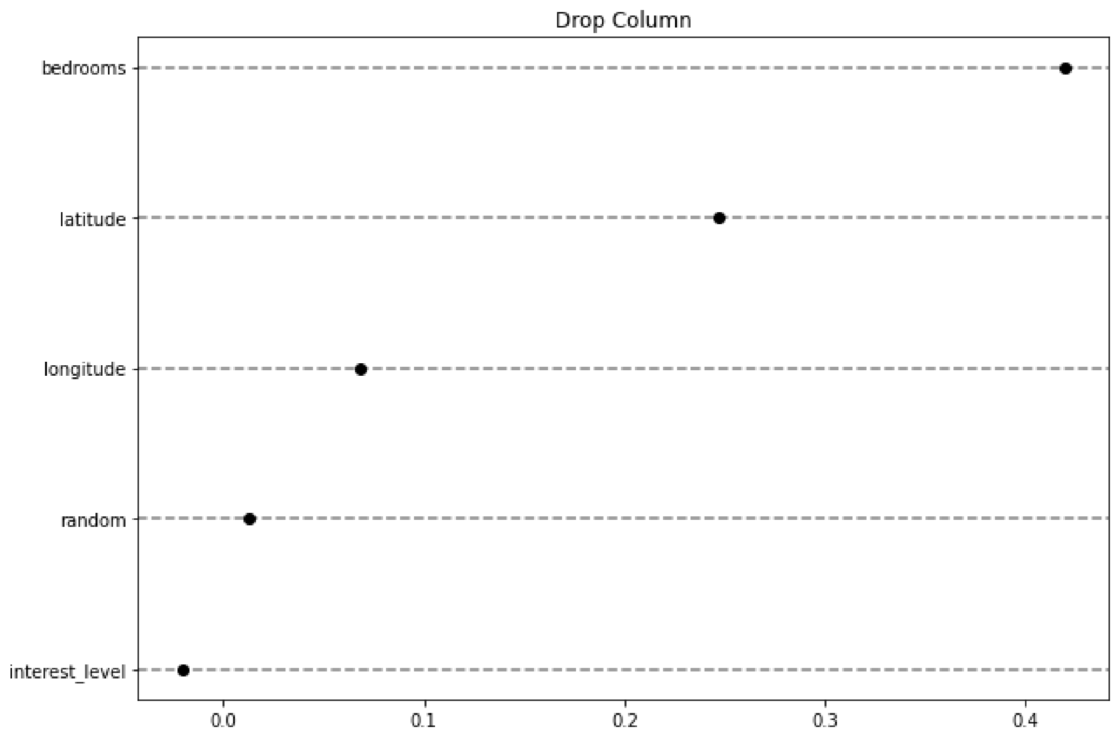
The example of permutation feature importance based on the house rent data is,

[6]:
```
rank = test_case_permutation()
plot_rank_top5(rank, "permutation")
```



And the example of drop-column feature importance based on the house rent data is,

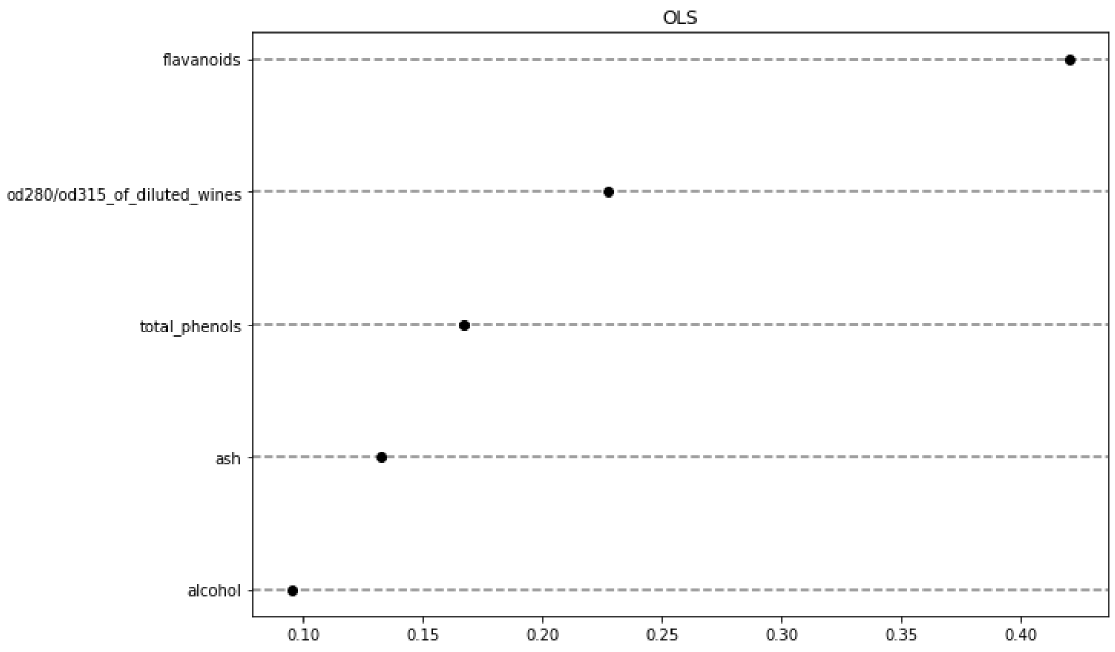
[7]:
```
rank = test_case_drop_column()
plot_rank_top5(rank, "Drop Column")
```



1.2.5 OLS Coefficients

A not so good feature importance is that we can first conduction a multivariate linear regression of features on y and then rank them based on the absolute value of their coefficients. This importance is not that good becuase we can only have the uniformed result if we normalized the features in the fiest place. However, for some categorical features, it makes no sense to normalize them before selection, which makes this technique limited in practice.

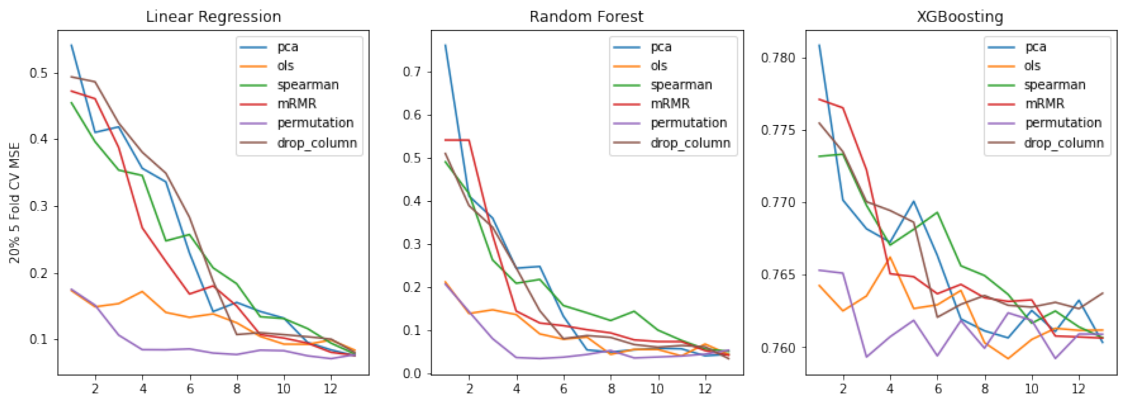
[8]:
```
rank = test_case_OLS()
plot_rank_top5(rank, "OLS")
```



1.2.6 Comparing Different Techniques

If we compare different techniques together, we can easily observe that different feature selection techniques yields to different results. In our results for linear regression, random forest, and gradient boosting with 5-fold cross validation, techniques of PCA, mRMR, drop-column shows a relatived strong ability in reducing the mean square error, and spearman rank correlcation is not that good compared with them, but permutation and OLS feature importance turns out to be relatively not well on finding the features that best reduce the mean square error.

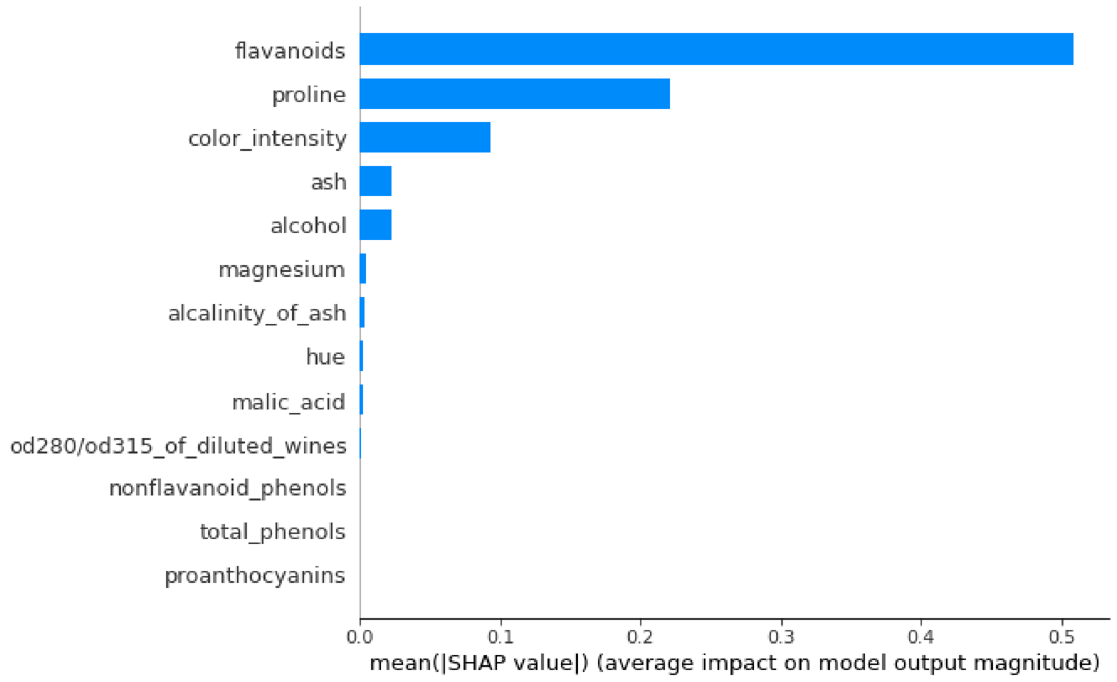
[6]:
```
warnings.simplefilter(action='ignore', category=FutureWarning)
plot_cv_scores()
```



1.2.7 Shap

Another technique called SHAP can also be used for feature selection, but we are not going to dive into that topic in this note. This is a package called shap and we will use that package for getting the SHAP feature importance. An example should be as follows.

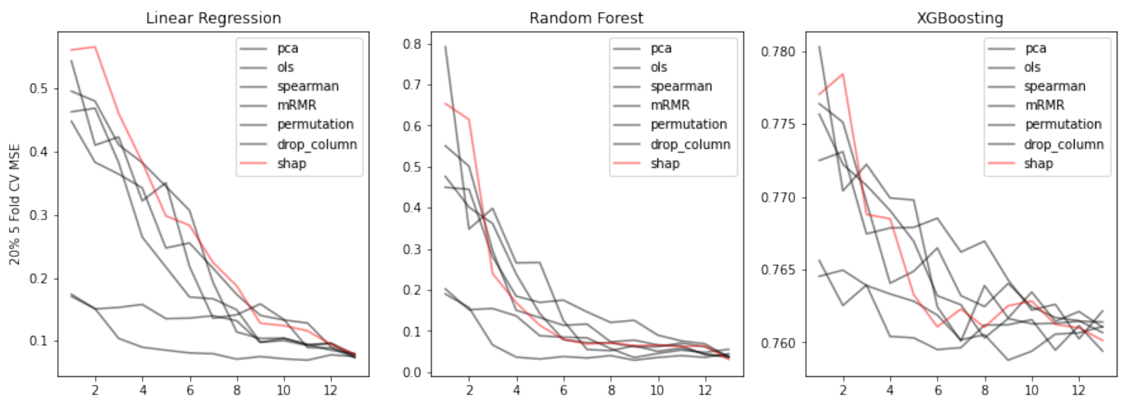
[2]:
```
plot_shap()
```



1.2.8 Comparing Techniques with Shap

Then finally in this part, we can use SHAP as a benchmark to tell which technique performs well. And the conclusion is quite similar. We can find out in this case that SHAP performs really well in feature selection.

[7]:
```
plot_cv_scores(shap=True)
```



1.3 Auto Feature Selection Algorithm

In this section, we are going to introduce an algorithm called auto feature selection (AFS). Consider a situation that we have 100,000 features and we don't know how many featrues we should select for building a good model, then how can we select the best number of remaining features k. The stepwise idea is as follows,

- Put all the features in the model and get a validation score
- Loop:
  - Use a festure selection technique to rank all the features
  - Drop the least important feature from out dataset
  - Retrain the model based on the new dataset with column dropped and get a validation score
  - Compare the current validation score with the last validation score we retrieve
    - ∗ If we have an absolute difference higher than our tolerance `tor`, we should stop
    - ∗ If we have an absolute difference lower than our tolerance `tor`, we should continue and drop more columns
- End loop.
- Get how many features we finally have, and let it be k.

Then, let's perform AFS to OLS feature selection and PCA feature selection as follows,

[2]:
```
auto_feature_selection("ols_score", tor=0.06)
```

```
[log] diff: 0.06705485714285714, # of removed features: 1
```

[2]:
```
(13,
 ['proline',
  'magnesium',
  'malic_acid',
  'hue',
  'alcalinity_of_ash',
  'color_intensity',
  'nonflavanoid_phenols',
  'proanthocyanins',
  'alcohol',
  'ash',
  'total_phenols',
  'od280/od315_of_diluted_wines',
  'flavanoids'])
```

[3]:
```
auto_feature_selection("pca_score", tor=0.06)
```

```
[log] diff: 0.037557396825396834, # of removed features: 1
[log] diff: 0.004637857142857131, # of removed features: 2
[log] diff: 0.004791698412698407, # of removed features: 3
[log] diff: 0.004308111111111103, # of removed features: 4
[log] diff: 0.003332968253968255, # of removed features: 5
[log] diff: 0.003851888888888877, # of removed features: 6
[log] diff: 0.023608174603174596, # of removed features: 7
[log] diff: 0.005144936507936508, # of removed features: 8
[log] diff: 0.0036752222222222325, # of removed features: 9
[log] diff: 0.012842301587301569, # of removed features: 10
[log] diff: 0.0272032380952381, # of removed features: 11
[log] diff: 0.1676884786366883, # of removed features: 12
```

[3]:
```
(2, ['od280/od315_of_diluted_wines', 'proline'])
```

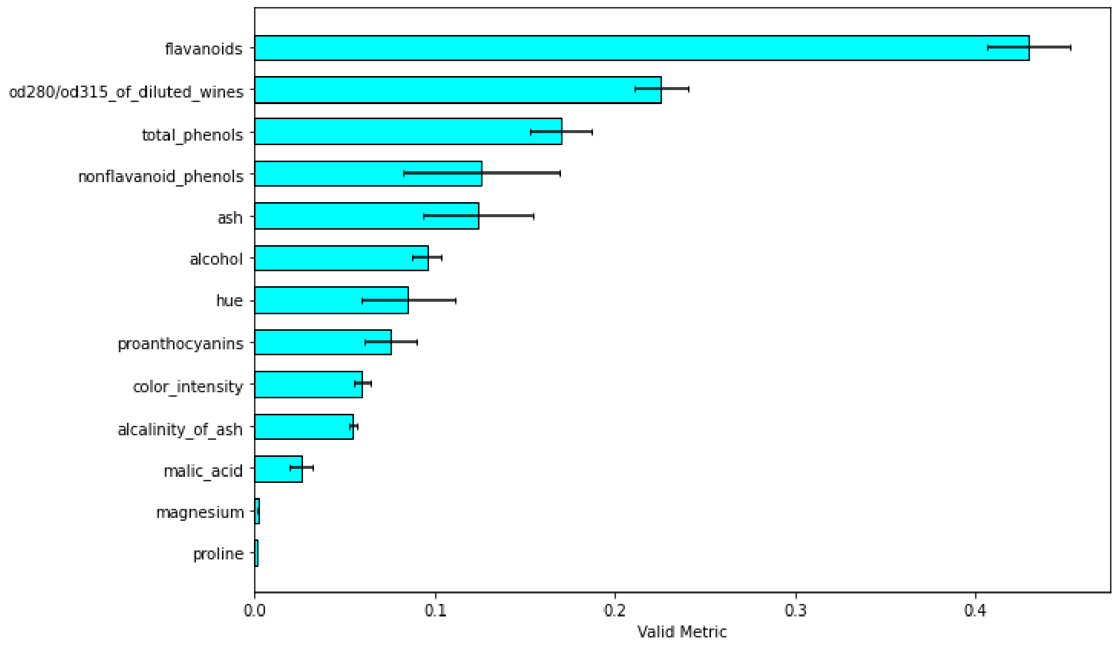
From the result above, we can tell that with the same tolerance of `0.06`, the PCA technique selects 2 features, but the OLS coefficient technique selects 13 features. This means that PCA is much more efficient than OLS coefficients for feature selection.

Note that in this case, we also implement the other feature selection techniques and you should refer to `featimp.py` for more information.

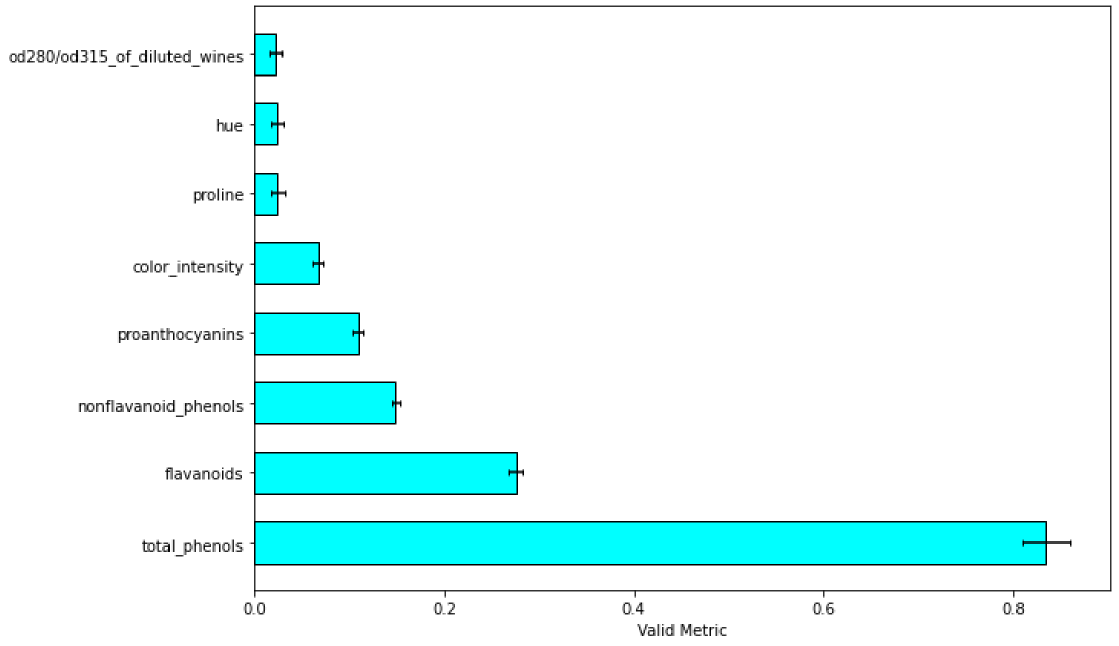
1.4 Confidence Interval for Feature Importance

We can also get the confidence interval for each feature by taking a random part of X and y (we take 80% with shuffling in our example) and then train by that partical data to get sightly different importances. Based on these importances for each feature, we are able to derive the confidence intervals for each feature. The example of OLS coefficients and PCA are as follows, and we also implement some other techniques which can be found in `featimp.py`.

[2]:
```
scores, feature_names = plot_importance_confidence("ols")
rank = get_confidence_intervals(scores, feature_names)
plot_confidence_interval(rank)
```



[3]:
```
scores, feature_names = plot_importance_confidence("pca")
rank = get_confidence_intervals(scores, feature_names)
rank.sort_values('mean', ascending=False, inplace=True)
plot_confidence_interval(rank.iloc[5:])
```


od280/od315_of_diluted_wines
hue
proline
color_intensity
proanthocyanins
nonflavanoid_phenols
flavanoids
total_phenols
Valid Metric