Lab #6: Feature Selection[1]

This lab is to deal with feature selection methods. These are important in improving the performance of machine learning algorithms.

==

Feature selection is a process where you automatically select those features in your data that contribute most to the prediction variable or output in which you are interested.

Having irrelevant features in your data can decrease the accuracy of many models, especially linear algorithms like linear and logistic regression.

Three benefits of performing feature selection before modeling your data are:

- **Reduces Overfitting**: Less redundant data means less opportunity to make decisions based on noise.
- **Improves Accuracy**: Less misleading data means modeling accuracy improves.
- **Reduces Training Time**: Less data means that algorithms train faster.

Selected methods for feature selection:

1. Univariate Selection

Statistical tests can be used to select those features that have the strongest relationship with the output variable. Univariate feature selection works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. Scikit-learn exposes feature selection routines as objects that implement the `transform` method:

- `SelectKBest` removes all but the highest scoring features
- `SelectPercentile` removes all but a user-specified highest scoring percentage of features
- using common univariate statistical tests for each feature: false positive rate `SelectFpr`, false discovery rate `SelectFdr`, or family wise error `SelectFwe`.
- `GenericUnivariateSelect` allows to perform univariate feature selection with a configurable strategy. This allows to select the best univariate selection strategy with hyper-parameter search estimator.

These objects take as input a scoring function that returns univariate scores and p-values (or only scores for ***SelectKBest*** and ***SelectPercentile***):

- For regression: `f_regression`, `mutual_info_regression`
- For classification: `chi2`, `f_classif`, `mutual_info_classif`

[1] https://scikit-learn.org/stable/modules/feature_selection.html

Usage:

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
X, y = load_iris(return_X_y=True)
X.shape

X_new = SelectKBest(chi2, k=2).fit_transform(X, y)
X_new.shape
```

2. Recursive Feature Elimination

The Recursive Feature Elimination (or RFE) 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.

Usage:

```
from sklearn.feature_selection import RFE
# feature extraction
model = LogisticRegression(solver='lbfgs')
rfe = RFE(model, 3)
fit = rfe.fit(X, y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

3. Principal Component Analysis

Principal Component Analysis (or PCA) uses linear algebra to transform the dataset into a compressed form. Generally this is called a data reduction technique. A property of PCA is that you can choose the number of dimensions or principal component in the transformed result.

Usage:

```
from sklearn.decomposition import PCA
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print(fit.components_)
```

4. Feature Importance

Bagged decision trees like Random Forest and Extra Trees can be used to estimate the importance of features.

Usage: Extra Trees

*Extremely Randomized Trees (also known as Extra Trees), construct multiple trees like Random Forest algorithms during training time over the entire dataset. Concretely, the Extra Trees will **construct trees over every observation in the dataset but with different subsets of features**.*

```
from sklearn.ensemble import ExtraTreesClassifier
# feature extraction
model = ExtraTreesClassifier(n_estimators=10)
model.fit(X, y)
print(model.feature_importances_)
```

Usage: RandomForest

*A Random Forest constructs multiple decision trees (a forest) during training time over different subsets of the training data. Concretely, an RF algorithm will **sample subsets of observations with different features from the dataset**. A decision tree is constructed over this subset.*

```
from sklearn.ensemble import RandomForestClassifier
#Create a Random Forest Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X,y)

#Report the importance values of features
pd.Series(clf.feature_importances_,index=dataset.feature_names).sort_va
lues(ascending=False)
```

The similarity between Random Forest and Extra Tree: They both construct multiple decision trees to use for the task at hand, whether classification or regression.

The difference between Random Forest and Extra Tree

Random Forest	Extremely Randomized Trees
Samples subsets through bootstrapping	Samples the entire dataset
Nodes are split looking at the best split	Randomized node split
Medium Variance	Low Variance
It takes time to find the best node to split on	Faster since node splits are random