FeatureSelection in Python

October 27, 2018

```
In [52]: import numpy as np
         import matplotlib.pyplot as plt
         from sklearn import datasets
```

1 Feature ranking

1.1 Remove low-variance features

```
In [53]: from sklearn.feature_selection import VarianceThreshold
         X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
         sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
         sel.fit_transform(X)
```

```
Out[53]: array([[0, 1],
                [1, 0],
                [0, 0],
                [1, 1],
                [1, 0],
                [1, 1]])
```

1.2 Classification scores

SelectKBest removes all but the highest scoring features. SelectPercentile removes all but a user-specified highest scoring percentage of features. SelectFpr/SelectFwe/SelectFdr univariate statistical tests for each feature: false positive rate Fpr, false discovery rate Fdr, or family wise error Fwe. GenericUnivariateSelect to perform univariate feature selection with a configurable strategy.

Classification scores: chi2, f_classif, mutual_info_classif

```
In [54]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import chi2

         iris = datasets.load_iris()
         X, y = iris.data, iris.target
         print(X.shape)

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

```
(150, 4)
(150, 2)
```

1.3 Regression scores

Regression scores: f_regression, mutual_info_regression

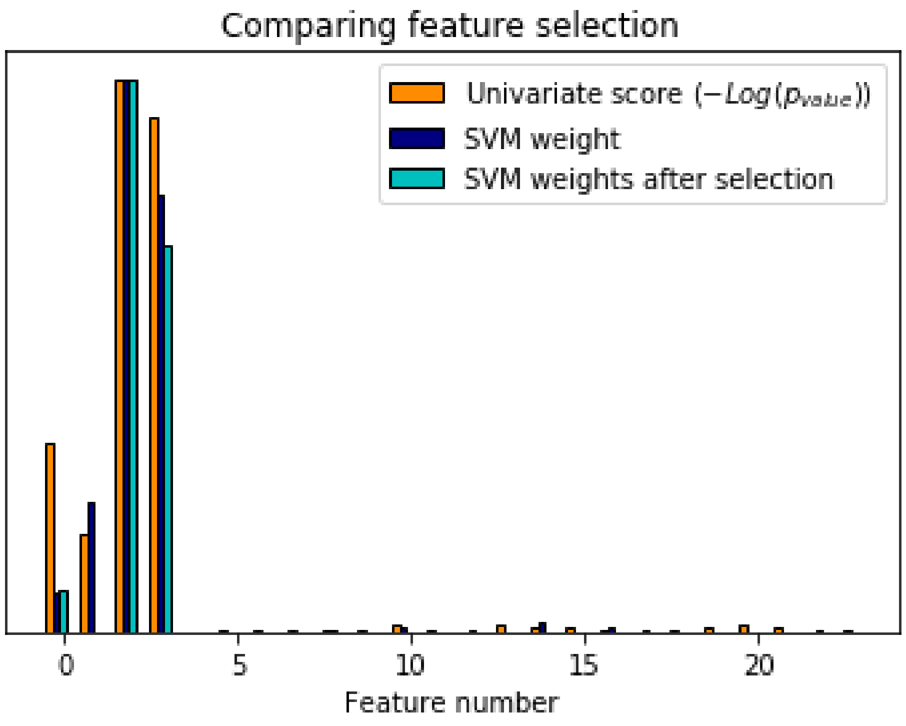
```
In [55]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import chi2

         # Load the boston dataset (regression)
         boston = datasets.load_boston()
         X, y = boston['data'], boston['target']
         print(X.shape)

         X_new = SelectPercentile(f_regression, percentile=30).fit_transform(X, y)
         print(X_new.shape)
```

```
(506, 13)
(506, 4)
```

1.4 F-test score

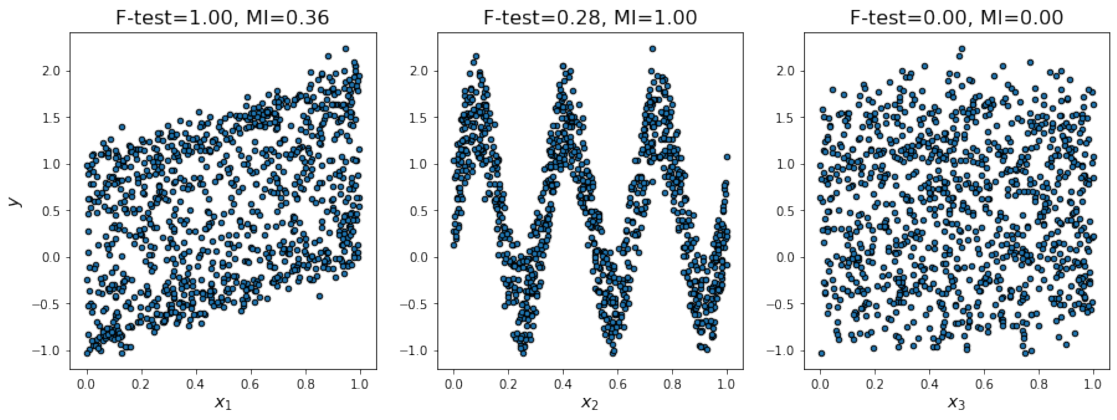
```
In [56]: from sklearn import datasets, svm
         from sklearn.feature_selection import SelectPercentile, f_classif

         # Add some noisy data to iris informative features
         iris = datasets.load_iris()
         E = np.random.uniform(0, 0.1, size=(len(iris.data), 20))
         X = np.hstack((iris.data, E))
         y = iris.target
         X_indices = np.arange(X.shape[-1])

         # Univariate feature selection with F-test for feature scoring
         selector = SelectPercentile(f_classif, percentile=10)
         selector.fit(X, y)

         scores = -np.log10(selector.pvalues_)
         scores /= scores.max()

         # Compare to the weights of an SVM
         clf = svm.SVC(kernel='linear')
         clf.fit(X, y)
         svm_weights = (clf.coef_ ** 2).sum(axis=0)
         svm_weights /= svm_weights.max()
```

```
clf_selected = svm.SVC(kernel='linear')
clf_selected.fit(selector.transform(X), y)

svm_weights_selected = (clf_selected.coef_ ** 2).sum(axis=0)
svm_weights_selected /= svm_weights_selected.max()

# PLOT
plt.figure(1)
plt.clf()
plt.bar(X_indices - .45, scores, width=.2,
        label=r'Univariate score ($-Log(p_{value})$)',
        color='darkorange', edgecolor='black')
plt.bar(X_indices - .25, svm_weights, width=.2, label='SVM weight',
        color='navy', edgecolor='black')
plt.bar(X_indices[selector.get_support()] - .05, svm_weights_selected,
        width=.2, label='SVM weights after selection', color='c', edgecolor='black')
plt.title("Comparing feature selection")
plt.xlabel('Feature number')
plt.yticks(())
plt.axis('tight')
plt.legend(loc='upper right')
plt.show()
```



1.5 Comparing ranking criteria

```
In [57]: #Comparison of F-test and mutual information
         from sklearn.feature_selection import f_regression, mutual_info_regression

         np.random.seed(0)
         X = np.random.rand(1000, 3)
         y = X[:, 0] + np.sin(6 * np.pi * X[:, 1]) + 0.1 * np.random.randn(1000)

         f_test, _ = f_regression(X, y)
         f_test /= np.max(f_test)

         mi = mutual_info_regression(X, y)
         mi /= np.max(mi)

         plt.figure(figsize=(15, 5))
         for i in range(3):
             plt.subplot(1, 3, i + 1)
             plt.scatter(X[:, i], y, edgecolor='black', s=20)
             plt.xlabel("$x_{}$".format(i + 1), fontsize=14)
             if i == 0: plt.ylabel("$y$", fontsize=14)
             plt.title("F-test={:.2f}, MI={:.2f}".format(f_test[i], mi[i]),fontsize=16)
         plt.show()
```



2 Subset feature selection

2.1 ANOVA filter

```
In [58]: from sklearn import svm
         from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import f_regression
         from sklearn.pipeline import Pipeline
```

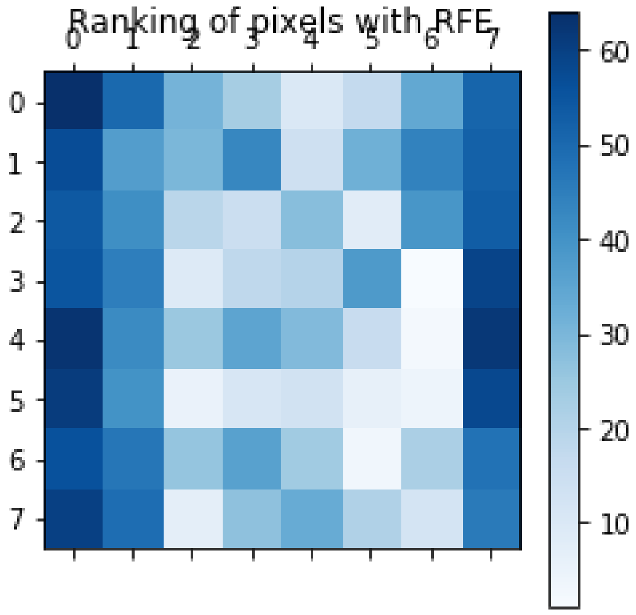
```
        # generate some data to play with
        X, y = datasets.samples_generator.make_classification(
                n_informative=5, n_redundant=0, random_state=42)

        # ANOVA using SVM classifier
        anova_filter = SelectKBest(f_regression, k=5)
        clf = svm.SVC(kernel='linear')
        anova_svm = Pipeline([('anova', anova_filter), ('svc', clf)])

        # You can set the parameters using the names issued
        # For instance, fit using a k of 10 in the SelectKBest and a parameter 'C' of the svm
        anova_svm.set_params(anova__k=10, svc__C=.1).fit(X, y)

        # SELECT FEATURES
        prediction = anova_svm.predict(X)
        print(anova_svm.score(X, y))

        # getting the selected features chosen by anova_filter
        print(anova_svm.named_steps['anova'].get_support())

        # Another way to get selected features chosen by anova_filter
        print(anova_svm.named_steps.anova.get_support())
```

```
0.83
[False False  True  True False False  True  True False  True False  True
  True False  True False  True  True False False]
[False False  True  True False False  True  True False  True False  True
  True False  True False  True  True False False]
```

2.2 Recursive feature elimination (RFE) filter

```
In [59]: from sklearn.svm import SVC
         from sklearn.datasets import load_digits
         from sklearn.feature_selection import RFE

         # Load the digits dataset
         digits = load_digits()
         X = digits.images.reshape((len(digits.images), -1))
         y = digits.target

         # Create the RFE object and rank each pixel
         svc = SVC(kernel="linear", C=1)
         rfe = RFE(estimator=svc, n_features_to_select=1, step=1)
         rfe.fit(X, y)
         ranking = rfe.ranking_.reshape(digits.images[0].shape)

         # Plot pixel ranking
```

```
plt.matshow(ranking, cmap=plt.cm.Blues)
plt.colorbar()
plt.title("Ranking of pixels with RFE")
plt.show()
```

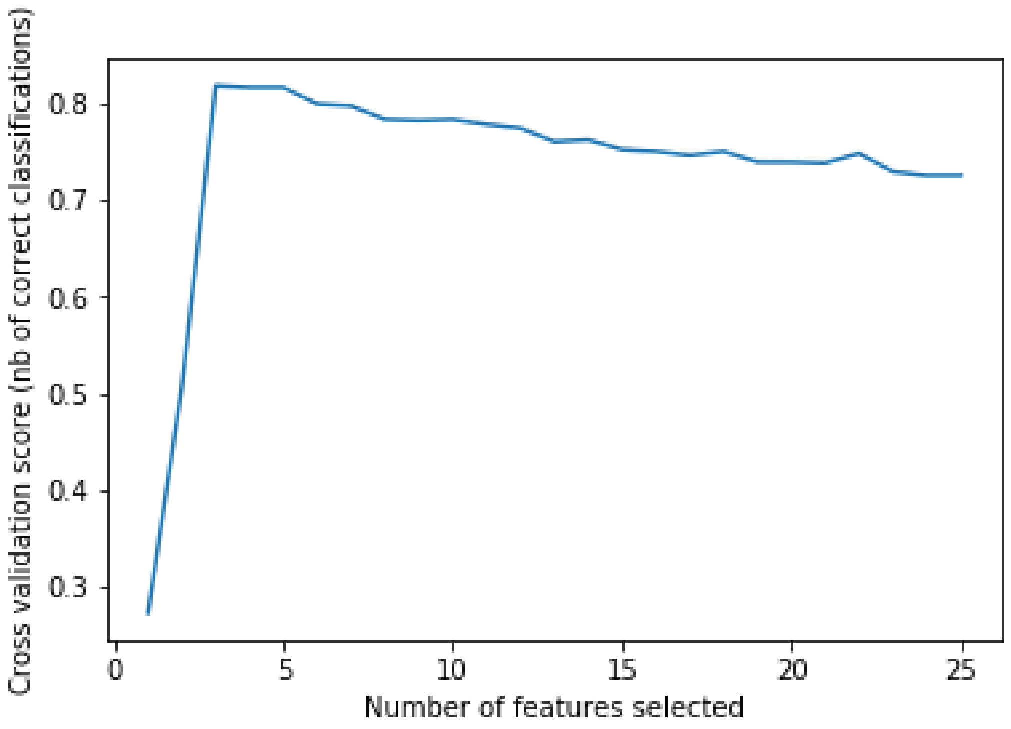


2.3 Wrapper using cross-validation

```
In [60]: #with cross-validation
         from sklearn.svm import SVC
         from sklearn.model_selection import StratifiedKFold
         from sklearn.feature_selection import RFECV
         from sklearn.datasets import make_classification

         # Build a classification task using 3 informative features
         X, y = make_classification(n_samples=1000, n_features=25, n_informative=3,
                                    n_redundant=2, n_repeated=0, n_classes=8,
                                    n_clusters_per_class=1, random_state=0)

         # Create the RFE object and compute a cross-validated score.
         svc = SVC(kernel="linear")
         # The "accuracy" scoring is proportional to the number of correct classifications
         rfecv = RFECV(estimator=svc, step=1, cv=StratifiedKFold(2), scoring='accuracy')
         rfecv.fit(X, y)
         print("Optimal number of features : %d" % rfecv.n_features_)

         # Plot number of features VS. cross-validation scores
```

```
        plt.figure()
        plt.xlabel("Number of features selected")
        plt.ylabel("Cross validation score (nb of correct classifications)")
        plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
        plt.show()
```

```
Optimal number of features : 3
```



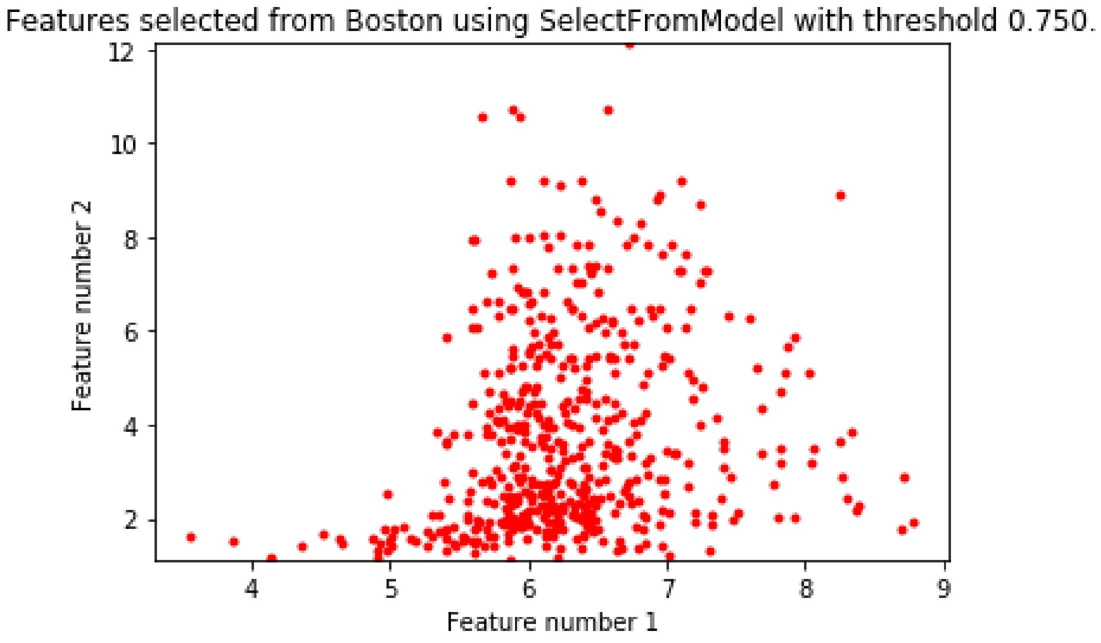
2.4 Other criteria (e.g. lasso CV, tree-based)

```
In [61]: #L1-based feature selection
         from sklearn.svm import LinearSVC
         from sklearn.feature_selection import SelectFromModel

         iris = datasets.load_iris()
         X, y = iris.data, iris.target
         print(X.shape)
         lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
         model = SelectFromModel(lsvc, prefit=True)
         X_new = model.transform(X)
         print(X_new.shape)
```

```
(150, 4)
(150, 3)
```

```
In [62]: from sklearn.feature_selection import SelectFromModel
         from sklearn.linear_model import LassoCV

         # Load the boston dataset.
         boston = datasets.load_boston()
         X, y = boston['data'], boston['target']
         print(X.shape[1])

         # We use the base estimator LassoCV since the L1 norm promotes sparsity of features.
         clf = LassoCV(cv=5)

         # Set a minimum threshold of 0.25
         sfm = SelectFromModel(clf, threshold=0.25)
         sfm.fit(X, y)
         n_features = sfm.transform(X).shape[1]
         print(n_features)

         # Reset the threshold till the number of features equals two
         while n_features > 2:
             sfm.threshold += 0.1
             X_transform = sfm.transform(X)
             n_features = X_transform.shape[1]

         # Plot the selected two features from X.
         plt.title(
             "Features selected from Boston using SelectFromModel with "
             "threshold %0.3f." % sfm.threshold)
         feature1 = X_transform[:, 0]
         feature2 = X_transform[:, 1]
         plt.plot(feature1, feature2, 'r.')
         plt.xlabel("Feature number 1")
         plt.ylabel("Feature number 2")
         plt.ylim([np.min(feature2), np.max(feature2)])
         plt.show()
```

```
13
5
```



```
In [63]: from sklearn.ensemble import ExtraTreesClassifier
         from sklearn.feature_selection import SelectFromModel

         iris = datasets.load_iris()
         X, y = iris.data, iris.target
         print(X.shape)

         #Tree-based feature selection with unknown number of features
         clf = ExtraTreesClassifier(n_estimators=50)
         clf = clf.fit(X, y)
         print(clf.feature_importances_)

         model = SelectFromModel(clf, prefit=True)
         X_new = model.transform(X)
         print(X_new.shape)
```

```
(150, 4)
[0.11447254 0.05427899 0.36703291 0.46421557]
(150, 2)
```