attribute_selection_reduced

December 18, 2015

1 ATTRIBUTE / FEATURE SELECTION

First, relevant libraries are imported

```
In [79]: from sklearn.datasets import load_boston
         from sklearn import tree
         from sklearn.cross_validation import train_test_split, cross_val_score, KFold
         from sklearn import metrics
         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.grid_search import GridSearchCV
         # Attribute selection methods from sklearn
         from sklearn.feature_selection import SelectKBest, SelectPercentile, chi2, f_classif, f_regres
```

1.1 RANKING/FILTER ATTRIBUTE SELECTION WITH TRAIN / TEST

```
In [80]: boston = load_boston()
         X, y = boston.data, boston.target
         print(X.shape)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_state=33)
```

```
(506L, 13L)
```

Let's see the estimated accuracy with the original dataset (all the attributes)

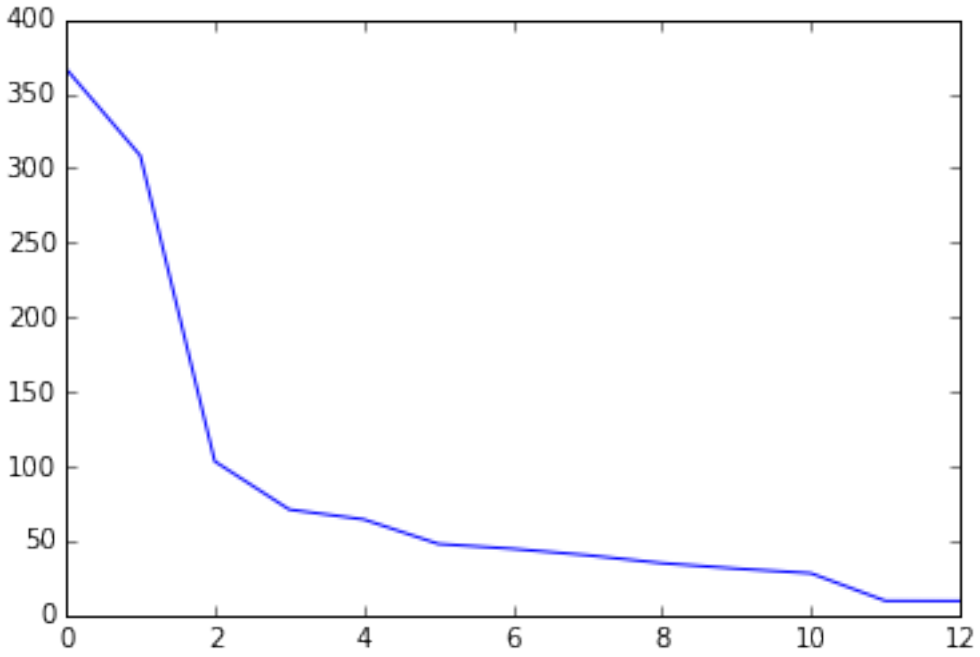
```
In [81]: clf = tree.DecisionTreeRegressor()
         clf = clf.fit(X_train, y_train)
         y_test_pred = clf.predict(X_test)
         print metrics.mean_squared_error(y_test, y_test_pred)
```

```
18.6822660099
```

```
In [82]: %matplotlib inline
         # We want to rank all attributes, and the best ones will be selected later
         selector = SelectKBest(f_regression, k="all")
         selector.fit(X_train, y_train)
         sorted_attributes = np.argsort(-selector.scores_)
         sorted_scores = np.sort(-selector.scores_)
         for index,element in enumerate(zip(sorted_attributes, sorted_scores)):
             print element
             if index>10: break

         plt.plot(-sorted_scores)
         plt.show()
```

```
(12, -366.83409486707029)
(5, -308.37535794972194)
(10, -103.27726686601574)
(2, -70.968775051003149)
(9, -64.349952462738642)
(4, -47.922568894311766)
(0, -44.628812695117624)
(1, -40.337404608424059)
(8, -35.025756985597887)
(6, -31.350893876051142)
(11, -28.25284632711735)
(7, -9.7600313040364668)
```



It seems that the first three attributes are the most correlated with the label. Let's see what happens if we select only the best three attributes.

```
In [83]: # Select the first 3 best attributes
         X_train_new = X_train[:, sorted_attributes[0:3]]
         X_test_new = X_test[:, sorted_attributes[0:3]]
```

We can see that the error is not too different, even though we have removed most of the attributes.

```
In [84]: clf = tree.DecisionTreeRegressor()
         clf = clf.fit(X_train, y_train)
         y_test_pred = clf.predict(X_test)
         print metrics.mean_squared_error(y_test, y_test_pred)
```

```
16.9882758621
```

Now, we are going to construct a method which is a combination or a sequence (a pipeline, in fact) of an attribute selector + a decision tree regressor. clf is therefore the pipeline (a sequence of attribute selection + regression algorithm). The number of attributes to be selected is a hyper-parameter of clf. max_depth is also a hyper-parameter of clf. We can use grid search in order to tune both parameters.

```
In [86]: from sklearn.pipeline import Pipeline

         param_grid = {'feature_selection__k': np.arange(X_train.shape[1])+1,
                       'regression__max_depth': np.arange(4)+1}

         clf = Pipeline([
           ('feature_selection', SelectKBest(f_regression)),
           ('regression', tree.DecisionTreeRegressor())
         ])

         clf_grid = GridSearchCV(clf,
                                 param_grid,
                                 scoring='mean_squared_error',
                                 cv=5 , n_jobs=1, verbose=1)
         %time _ = clf_grid.fit(X,y)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks       | elapsed:    0.2s
[Parallel(n_jobs=1)]: Done 199 tasks       | elapsed:    1.0s

Fitting 5 folds for each of 52 candidates, totalling 260 fits
Wall time: 1.51 s

[Parallel(n_jobs=1)]: Done 260 out of 260 | elapsed:    1.4s finished
```

Now, let's see the best hyper-parameters. It seems that for this case, 3 attributes should be selected

```
In [89]: print clf_grid.best_params_, clf_grid.best_score_
```

```
{'feature_selection__k': 3, 'regression__max_depth': 3} -30.311835006
```

```
Out[89]: [mean: -30.31184, std: 16.85403, params: {'feature_selection__k': 3, 'regression__max_depth': 3}
          mean: -30.39138, std: 17.51927, params: {'feature_selection__k': 2, 'regression__max_depth': 3}
          mean: -30.43885, std: 18.08344, params: {'feature_selection__k': 3, 'regression__max_depth': 4}
          mean: -31.41281, std: 15.55386, params: {'feature_selection__k': 7, 'regression__max_depth': 3}
          mean: -31.66207, std: 15.95840, params: {'feature_selection__k': 5, 'regression__max_depth': 3}
          mean: -32.31163, std: 15.17629, params: {'feature_selection__k': 6, 'regression__max_depth': 3}
          mean: -32.63583, std: 15.19363, params: {'feature_selection__k': 4, 'regression__max_depth': 3}
          mean: -32.84554, std: 21.35249, params: {'feature_selection__k': 2, 'regression__max_depth': 4}
          mean: -33.93441, std: 11.92595, params: {'feature_selection__k': 7, 'regression__max_depth': 4}
          mean: -36.48472, std: 13.54363, params: {'feature_selection__k': 13, 'regression__max_depth': 3
```

```
In [90]: clf_grid.grid_scores_.sort(key=lambda(x): -x[1])
         clf_grid.grid_scores_[0:10]
```

```
Out[90]: [mean: -30.31184, std: 16.85403, params: {'feature_selection__k': 3, 'regression__max_depth': 3}
          mean: -30.39138, std: 17.51927, params: {'feature_selection__k': 2, 'regression__max_depth': 3}
          mean: -30.43885, std: 18.08344, params: {'feature_selection__k': 3, 'regression__max_depth': 4}
          mean: -31.41281, std: 15.55386, params: {'feature_selection__k': 7, 'regression__max_depth': 3}
          mean: -31.66207, std: 15.95840, params: {'feature_selection__k': 5, 'regression__max_depth': 3}
          mean: -32.31163, std: 15.17629, params: {'feature_selection__k': 6, 'regression__max_depth': 3}
          mean: -32.63583, std: 15.19363, params: {'feature_selection__k': 4, 'regression__max_depth': 3}
          mean: -32.84554, std: 21.35249, params: {'feature_selection__k': 2, 'regression__max_depth': 4}
          mean: -33.93441, std: 11.92595, params: {'feature_selection__k': 7, 'regression__max_depth': 4}
          mean: -36.48472, std: 13.54363, params: {'feature_selection__k': 13, 'regression__max_depth': 3
```

2 USING PCA FOR TRANSFORMING ATTRIBUTES WITH TRAIN / TEST EVALUATION

```
In [107]: from sklearn import decomposition
          from sklearn import datasets

In [108]: iris = datasets.load_iris()
          X = iris.data
          y = iris.target
          print X.shape
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_state=33)
```

```
(150L, 4L)
```

Let's use the maximum number of PCA components for the moment(4 iris attributes implies 4 attributes)

```
In [114]: pca = decomposition.PCA(n_components=4)
          pca.fit(X_train)
```

```
Out[114]: PCA(copy=True, n_components=4, whiten=False)
```

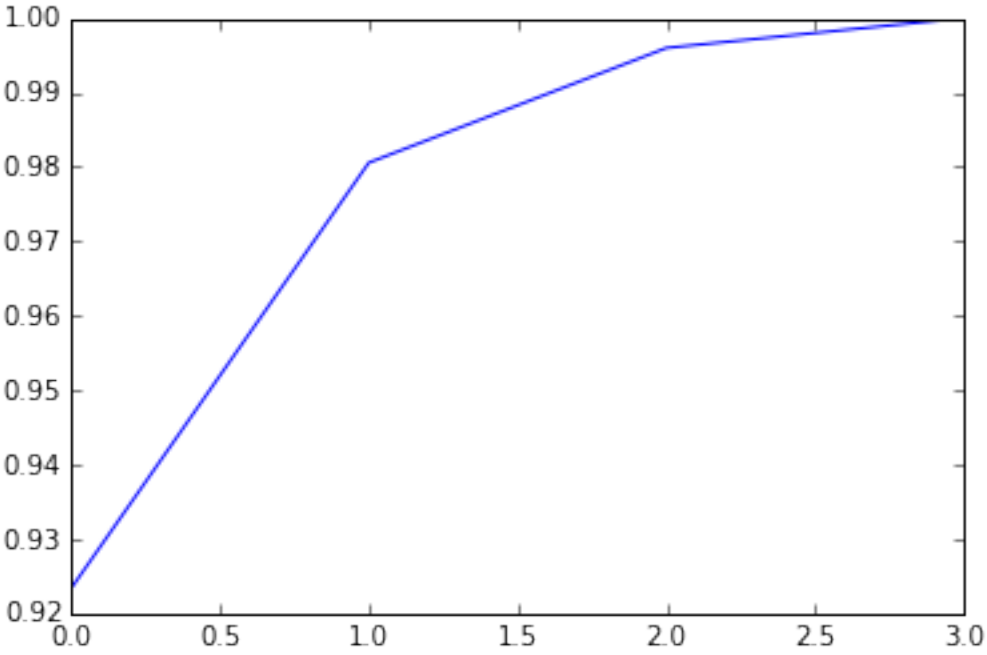
Now, let's see how much variance explains each of the four components. We can see that the first component explaines most of the variance 92%

```
In [119]: pca.explained_variance_ratio_
```

```
Out[119]: array([ 0.92337895,  0.05717063,  0.01542177,  0.00402865])
```

Now, let's compute the cummulative variance explained by n components. It seems that with 2 components we can already explain more than 95% of the variance. Using that criterion, we should use 2 components.

```
In [122]: accumulated_variance = np.cumsum(pca.explained_variance_ratio_)
          plt.plot(accumulated_variance)
          plt.show()
```



Therefore, let's compute 2 PCA components and apply them to train and test. We can see that the new input attributes (X_train_new and X_test_new have 2 new attributes)

```
In [125]: pca = decomposition.PCA(n_components=2)
          pca.fit(X_train)
          X_train_new = pca.transform(X_train)
          X_test_new = pca.transform(X_test)
```

Now, we can apply a classifier to the new, reduced, training set, and test it on the transformed test set

```
In [130]: clf = tree.DecisionTreeClassifier()
          clf = clf.fit(X_train_new, y_train)
```

```
In [136]: y_train_pred = clf.predict(X_train_new)
          y_test_pred = clf.predict(X_test_new)
          print metrics.accuracy_score(y_train, y_train_pred)
          print metrics.accuracy_score(y_test, y_test_pred)
```

```
1.0
0.916666666667
```

If we construct the tree with the original dataset (before applying PCA), we see that test accuracy is larger with 4 attributes than with 2 PCA components. So in this case, PCA would not be useful from an accuracy point of view, but it would be useful to reduce the complexity of the model (with PCA we have only 2 components instead of the 4 original attributes).

```
In [138]: clf = tree.DecisionTreeClassifier()
          clf = clf.fit(X_train, y_train)
          y_train_pred = clf.predict(X_train)
          y_test_pred = clf.predict(X_test)
          print metrics.accuracy_score(y_train, y_train_pred)
          print metrics.accuracy_score(y_test, y_test_pred)
```

```
1.0
0.95
```

```
In [ ]:
```