

特征选择

当你创建了成千上万个特征后，就该从中挑选出几个了。但是，我们绝不应该创建成百上千个无用的特征。特征过多会带来一个众所周知的问题，即 "维度诅咒"。如果你有很多特征，你也必须有很多训练样本来捕捉所有特征。什么是 "大量" 并没有正确的定义，这需要您通过正确验证您的模型和检查训练模型所需的时间来确定。

选择特征的最简单方法是**删除方差非常小的特征**。如果特征的方差非常小（即非常接近于 0），它们就接近于常量，因此根本不会给任何模型增加任何价值。最好的办法就是去掉它们，从而降低复杂度。请注意，方差也取决于数据的缩放。Scikit-learn 的 `VarianceThreshold` 实现了这一点。

```
from sklearn.feature_selection import VarianceThreshold
data = ...
var_thresh = VarianceThreshold(threshold=0.1)
transformed_data = var_thresh.fit_transform(data)
```

我们还可以删除相关性较高的特征。要计算不同数字特征之间的相关性，可以使用皮尔逊相关性。

```
import pandas as pd
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing()
X = data["data"]
col_names = data["feature_names"]
y = data["target"]
df = pd.DataFrame(X, columns=col_names)
df.loc[:, "MedInc_Sqrt"] = df.MedInc.apply(np.sqrt)
df.corr()
```

得出相关矩阵，如图 1 所示。

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedInc_Sqrt
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.079809	-0.015176	0.984329
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.011173	-0.108197	-0.132797
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.106389	-0.027540	0.326688
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.069721	0.013344	-0.066910
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.108785	0.099773	0.018415
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.002366	0.002476	0.015266
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.000000	-0.924664	-0.084303
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.924664	1.000000	-0.015569
MedInc_Sqrt	0.984329	-0.132797	0.326688	-0.066910	0.018415	0.015266	-0.084303	-0.015569	1.000000

图 1：皮尔逊相关矩阵样本

我们看到，MedInc_Sqrt 与 MedInc 的相关性非常高。因此，我们可以删除其中一个特征。

现在我们可以转向一些[单变量特征选择方法](#)。单变量特征选择只不过是针对给定目标对每个特征进行评分。[互信息](#)、[方差分析 F 检验](#)和 [chi2](#) 是一些最常用的单变量特征选择方法。在 scikit-learn 中，有两种方法可以使用这些方法。

- SelectKBest：保留得分最高的 k 个特征
- SelectPercentile：保留用户指定百分比内的顶级特征。

必须注意的是，只有非负数据才能使用 chi2。在自然语言处理中，当我们有一些单词或基于 tf-idf 的特征时，这是一种特别有用的特征选择技术。最好为单变量特征选择创建一个包装器，几乎可以用于任何新问题。

```
from sklearn.feature_selection import chi2
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import SelectPercentile
class UnivariateFeatureSelction:
    def __init__(self, n_features, problem_type, scoring):
        if problem_type == "classification":
            valid_scoring = {
                "f_classif": f_classif,
                "chi2": chi2,
                "mutual_info_classif": mutual_info_classif
            }
        else:
            valid_scoring = {
                "f_regression": f_regression,
                "mutual_info_regression": mutual_info_regression
```

```

    }
    if scoring not in valid_scoring:
        raise Exception("Invalid scoring function")
    if isinstance(n_features, int):
        self.selection = SelectKBest(
            valid_scoring[scoring],
            k=n_features
        )
    elif isinstance(n_features, float):
        self.selection = SelectPercentile(
            valid_scoring[scoring],
            percentile=int(n_features * 100)
        )
    else:
        raise Exception("Invalid type of feature")

    def fit(self, X, y):
        return self.selection.fit(X, y)
    def transform(self, X):
        return self.selection.transform(X)
    def fit_transform(self, X, y):
        return self.selection.fit_transform(X, y)

```

使用该类非常简单。

```

ufs = UnivariateFeatureSelction(n_features=0.1,
                                problem_type="regression",
                                scoring="f_regression"
                                )

ufs.fit(X, y)
X_transformed = ufs.transform(X)

```

这样就能满足大部分单变量特征选择的需求。请注意，创建较少而重要的特征通常比创建数以百计的特征要好。单变量特征选择不一定总是表现良好。大多数情况下，人们更喜欢使用机器学习模型进行特征选择。让我们来看看如何做到这一点。

使用模型进行特征选择的最简单形式被称为贪婪特征选择。在贪婪特征选择中，第一步是选择一个模型。第二步是选择损失/评分函数。第三步也是最后一步是反复评估每个特征，如果能提高损失/评分，就将其添加到 "好" 特征列表中。没有比这更简单的了。但你必须记住，这被称为贪婪特征选择是有原因的。这种特征选择过程在每次评估特征时都会适合给定的模型。这种方法的计算成本非常高。完成这种特征选择也需要大量时间。如果不正确使用这种特征选择，甚至会导致模型过度拟合。

让我们来看看它是如何实现的。

```
import pandas as pd
from sklearn import linear_model
from sklearn import metrics
from sklearn.datasets import make_classification

class GreedyFeatureSelection:
    def evaluate_score(self, X, y):
        model = linear_model.LogisticRegression()
        model.fit(X, y)
        predictions = model.predict_proba(X)[:, 1]
        auc = metrics.roc_auc_score(y, predictions)
        return auc

    def _feature_selection(self, X, y):
        good_features = []
        best_scores = []
        num_features = X.shape[1]
        while True:
            this_feature = None
            best_score = 0

            for feature in range(num_features):
                if feature in good_features:
                    continue
                selected_features = good_features + [feature]
                xtrain = X[:, selected_features]
                score = self.evaluate_score(xtrain, y)
                if score > best_score:
                    this_feature = feature
                    best_score = score
            if this_feature != None:
                good_features.append(this_feature)
                best_scores.append(best_score)
            if len(best_scores) > 2:
                if best_scores[-1] < best_scores[-2]:
                    break
        return best_scores[:-1], good_features[:-1]

    def __call__(self, X, y):
        scores, features = self._feature_selection(X, y)
        return X[:, features], scores

if __name__ == "__main__":
    X, y = make_classification(n_samples=1000, n_features=100)
    X_transformed, scores = GreedyFeatureSelection()(X, y)
```

这种贪婪特征选择方法会返回分数和特征索引列表。图 2 显示了在每次迭代中增加一个新特征后，分数是如何提高的。我们可以看到，在某一点之后，我们就无法提高分数了，这就是我们停止的地方。

另一种贪婪的方法被称为递归特征消除法（RFE）。在前一种方法中，我们从一个特征开始，然后不断添加新的特征，但在 RFE 中，我们从所有特征开始，在每次迭代中不断去除一个对给定模型提供最小值的特征。但我们如何知道哪个特征的价值最小呢？如果我们使用线性支持向量机（SVM）或逻辑回归等模型，我们会为每个特征得到一个系数，该系数决定了特征的重要性。而对于任何基于树的模型，我们得到的是特征重要性，而不是系数。在每次迭代中，我们都可以剔除最不重要的特征，直到达到所需的特征数量为止。因此，我们可以决定要保留多少特征。

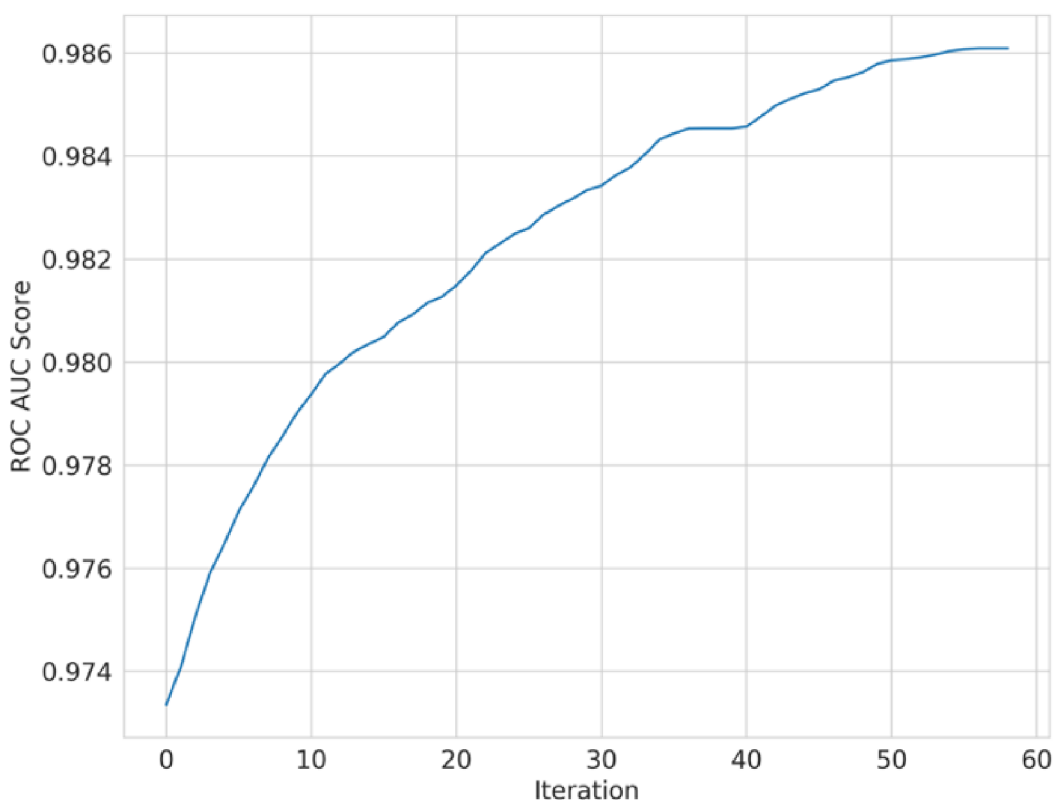


图 2：增加新特征后，贪婪特征选择的 AUC 分数如何变化

当我们进行递归特征剔除时，在每次迭代中，我们都会剔除特征重要性较高的特征或系数接近 0 的特征。请记住，当你使用逻辑回归这样的模型进行二元分类时，如果特征对正分类很重要，其系数就会更正，而如果特征对负分类很重要，其系数就会更负。修改我们的贪婪特征选择类，创建一个新的递归特征消除类非常容易，但 scikit-learn 也提供了 RFE。下面的示例展示了一个简单的用法。

```
import pandas as pd
```

```

from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing()
X = data["data"]
col_names = data["feature_names"]
y = data["target"]
model = LinearRegression()
rfe = RFE(
    estimator=model,
    n_features_to_select=3
)
rfe.fit(X, y)
X_transformed = rfe.transform(X)

```

我们看到了从模型中选择特征的两种不同的贪婪方法。但也可以根据数据拟合模型，然后通过特征系数或特征的重要性从模型中选择特征。如果使用系数，则可以选择一个阈值，如果系数高于该阈值，则可以保留该特征，否则将其剔除。

让我们看看如何从随机森林这样的模型中获取特征重要性。

```

import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.ensemble import RandomForestRegressor
data = load_diabetes()
X = data["data"]
col_names = data["feature_names"]
y = data["target"]
model = RandomForestRegressor()
model.fit(X, y)

```

随机森林（或任何模型）的特征重要性可按如下方式绘制。

```

importances = model.feature_importances_
idxs = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(idxs)), importances[idxs], align='center')
plt.yticks(range(len(idxs)), [col_names[i] for i in idxs])
plt.xlabel('Random Forest Feature Importance')
plt.show()

```

结果如图 3 所示。

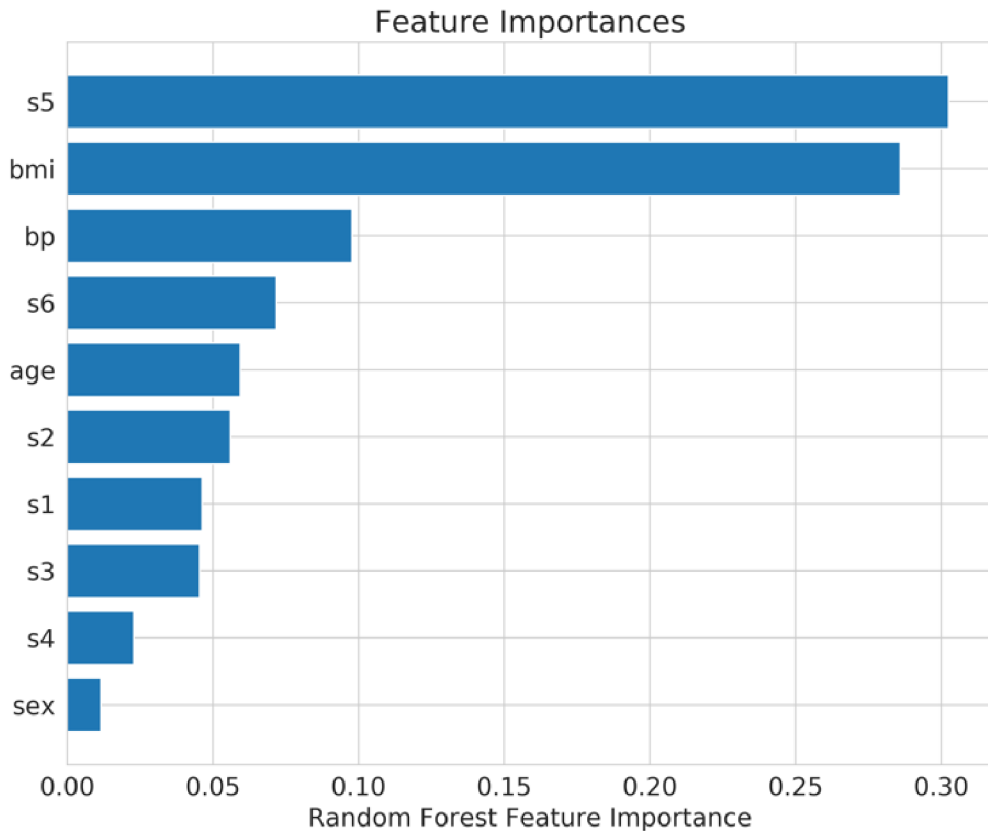


图 3：特征重要性图

从模型中选择最佳特征并不是什么新鲜事。您可以从一个模型中选择特征，然后使用另一个模型进行训练。例如，您可以使用逻辑回归系数来选择特征，然后使用随机森林（Random Forest）对所选特征进行模型训练。Scikit-learn 还提供了 `SelectFromModel` 类，可以帮助你直接从给定的模型中选择特征。您还可以根据需要指定系数或特征重要性的阈值，以及要选择的特征的最大数量。

请看下面的代码段，我们使用 `SelectFromModel` 中的默认参数来选择特征。

```
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel
data = load_diabetes()
X = data["data"]
col_names = data["feature_names"]
y = data["target"]
model = RandomForestRegressor()
sfm = SelectFromModel(estimator=model)
X_transformed = sfm.fit_transform(X, y)
support = sfm.get_support()
print([x for x, y in zip(col_names, support) if y == True ])
```

上面程序打印结果：['bmi', 's5']。我们再看图 3，就会发现这是最重要的两个特征。因此，我们也可以直接从随机森林提供的特征重要性中进行选择。我们还缺少一件事，那就是使用 **L1（Lasso）惩罚模型** 进行特征选择。当我们使用 L1 惩罚进行正则化时，大部分系数都将为 0（或接近 0），因此我们要选择系数不为 0 的特征。只需将模型选择片段中的随机森林替换为支持 L1 惩罚的模型（如 lasso 回归）即可。所有基于树的模型都提供特征重要性，因此本章中展示的所有基于模型的片段都可用于 XGBoost、LightGBM 或 CatBoost。特征重要性函数的名称可能不同，产生结果的格式也可能不同，但用法是一样的。最后，在进行特征选择时必须小心谨慎。在训练数据上选择特征，并在验证数据上验证模型，以便在不过度拟合模型的情况下正确选择特征。