回归问题

1、线性回归

线性回归通常是人们学习机器学习和数据科学的第一个算法。线性回归是一种线性模型，它假设输入变量 (X) 和单个输出变量 (y) 之间存在线性关系。 一般来说，有两种情况：

- 单变量线性回归：它对单个输入变量（单个特征变量）和单个输出变量之间的关系进行建模。
- 多变量线性回归（也称为多元线性回归）：它对多个输入变量（多个特征变量）和单个输出变量之间的关系进行建模。

Scikit-learn 在 LinearRegression() 中内置了这个功能。 让我们创建一个 LinearRegression 对象并将其拟合到训练数据中：

```
from sklearn.linear_model import LinearRegression
# Creating and Training the Model
linear_regressor = LinearRegression()
linear_regressor.fit(X, y)
```

关于线性回归的几个关键点：

- 快速且易于建模
- 当要建模的关系不是非常复杂并且您没有大量数据时，它特别有用。
- 非常直观的理解和解释。
- 它对异常值非常敏感。

2、多项式回归

当我们想要为非线性可分数据创建模型时，多项式回归是最受欢迎的选择之一。它类似于线性回归，但使用变量 X 和 y 之间的关系来找到绘制适合数据点的曲线的最佳方法。

Scikit-learn 的 PolynomialFeatures 内置了这种方法：

```
from sklearn.preprocessing import PolynomialFeatures
# We are simply generating the matrix for a quadratic model
poly_reg = PolynomialFeatures(degree = 2) #degree是多项式特征的最高次数
X_poly = poly_reg.fit_transform(X)
```

关于多项式回归的几个关键点：

- 能够对非线性可分数据进行建模；线性回归不能做到这一点。一般来说，它更加灵活，可以对一些相当复杂的关系进行建模。
- 完全控制特征变量的建模（要设置的指数）。
- 需要精心设计。 需要一些数据知识才能选择最佳指数。
- 如果指数选择不当，则容易过度拟合。

3、 支持向量机回归

持向量机在分类问题中是众所周知的。SVM 在回归中的使用称为支持向量回归(SVR)。Scikit-learn在SVR()中内置了这种方法。

在拟合 SVR 模型之前，通常最好的做法是执行特征缩放，以便每个特征具有相似的重要性。 首先，让我们使用 StandardScaler() 执行特征缩放：

```
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
# Performing feature scaling
scaled_X = StandardScaler()
scaled_y = StandardScaler()
scaled_X = scaled_X.fit_transform(X)
scaled_y = scaled_y.fit_transform(y)
```

接下来，我们创建一个 SVR 对象，内核设置为“rbf”，伽马设置为“auto”。 之后，我们调用 fit() 使其适合缩放的训练数据：

```
svr_regressor = SVR(kernel='rbf', gamma='auto')
svr_regressor.fit(scaled_X, scaled_y.ravel())
```

支持向量回归（Support Vector Regression，SVR）是一种基于支持向量机（Support Vector Machine，SVM）的回归方法。它与传统的线性回归和多项式回归相比具有一些优点和缺点，下面是对其进行总结：

优点：

1. 强大的非线性建模能力：SVR可以通过使用核函数将特征映射到高维空间，从而捕捉到复杂的非线性关系。这使得SVR在处理非线性回归问题时表现出色。
2. 鲁棒性：SVR使用支持向量作为关键训练样本，这些样本对模型的训练和预测起着重要的作用。由于只有少数支持向量被选中，SVR对于噪声和离群值的影响较小，具有较好的鲁棒性。
3. 控制模型的复杂度：通过调整正则化参数C和核函数的参数，可以控制SVR模型的复杂度。这使得SVR在避免过拟合和调整模型的泛化能力方面具有灵活性。
4. 数学基础和理论支持：SVR建立在支持向量机的理论基础上，有坚实的数学和统计学支持，具有较好的理论解释性。

缺点：

1. 计算复杂度高：SVR的计算复杂度随着训练样本数量的增加而增加，尤其是在使用核函数进行高维映射时。对于大规模数据集，训练时间可能会很长。
2. 参数选择敏感：SVR的性能受到正则化参数C和核函数参数的影响。选择合适的参数需要进行交叉验证或使用其他调参方法，这可能需要耗费一定的时间和计算资源。
3. 预测结果的解释性较差：由于SVR使用核函数进行特征映射，得到的模型是在高维空间中进行预测的。这使得预测结果的解释性较差，难以直观理解模型对于输入特征的影响。

4、决策树回归

决策树是一种用于分类和回归的非参数监督学习方法。目标是创建一个模型，通过学习从数据特征推断出的简单决策规则来预测目标变量的值。 一棵树可以看作是一个分段常数近似。决策树回归也很常见，以至于 Scikit-learn 将它内置在 DecisionTreeRegressor 中,可以在没有特征缩放的情况下创建 DecisionTreeRegressor 对象，（内置的决策树算法是**基于CART（Classification and Regression Trees）**算法的实现）如下所示：

```
from sklearn.tree import DecisionTreeRegressor
tree_regressor = DecisionTreeRegressor(random_state = 0)
tree_regressor.fit(X, y)
```

决策树回归（Decision Tree Regression）是一种基于决策树算法的回归方法。与传统的线性回归和多项式回归相比，决策树回归具有以下优点和缺点：

优点：

1. 简单直观：决策树模型易于理解和解释。它们可以生成清晰的决策规则，使得模型的输出结果具有直观的可解释性。
2. 非参数化模型：决策树回归不依赖于对数据分布的假设，因此适用于各种类型的数据（连续型和离散型特征）和不同分布的数据集。
3. 能够处理非线性关系：决策树可以自由地捕捉特征之间的非线性关系，而不需要进行特征转换或多项式扩展。
4. 处理离群值和缺失值能力强：决策树对于离群值和缺失值相对不敏感，在构建树时可以有效地处理这些问题。

缺点：

1. 容易过拟合：决策树有很高的灵活性，容易生成复杂的树结构，导致过拟合问题。为了解决过拟合，需要进行剪枝等模型调优技术。
2. 不稳定性：决策树对于数据的微小变化非常敏感，即使数据发生轻微的变化，生成的树结构也可能完全不同，这可能导致模型的不稳定性。
3. 忽略特征间的相关性：决策树在每个节点上只考虑一个特征进行划分，因此可能忽略了特征之间的相关性，无法充分利用特征之间的相互作用。
4. 可能产生高方差模型：由于决策树的自由生长和对数据的高度敏感性，很容易生成高方差的模型，对于噪声数据或局部细节过于敏感。

5、随机森林回归

随机森林回归基本上与决策树回归非常相似。 它是一个元估计器，可以在数据集的各种子样本上拟合多个决策树，并使用平均来提高预测准确性和控制过拟合。

随机森林回归器在回归中可能会或可能不会比决策树表现更好（虽然它通常在分类中表现更好），因为树构造算法本质上存在微妙的过拟合 - 欠拟合权衡。

随机森林回归很常见，以至于 Scikit-learn 将它内置在 RandomForestRegressor 中。 首先，我们需要创建一个具有指定数量的估计器的 RandomForestRegressor 对象，如下所示：

```
from sklearn.ensemble import RandomForestRegressor
forest_regressor = RandomForestRegressor(
    n_estimators = 300,
    random_state = 0
)
forest_regressor.fit(X, y.ravel())
```

随机森林回归具有以下特点和优点：

1. 高准确性：随机森林能够通过组合多个决策树模型来减少过拟合，提高模型的准确性和泛化能力。
2. 鲁棒性：随机森林对于噪声和离群值相对较强，可以处理具有噪声的数据集，不容易受到异常值的影响。
3. 可解释性：随机森林提供了特征的重要性排序，可以用于特征选择和解释模型对于输入特征的影响。
4. 并行化处理：由于每个决策树可以独立训练，随机森林可以通过并行化处理来加速模型的训练过程。

然而，随机森林回归也有一些注意事项：

1. 计算复杂度：随机森林在训练和预测阶段需要构建多个决策树模型，可能会消耗较多的计算资源和时间。
2. 参数调优：随机森林有一些重要的参数需要调优，例如决策树的数量、特征子集的大小等，需要通过交叉验证等方法来选择最佳的参数设置。

6、LASSO 回归

LASSO（Least Absolute Shrinkage and Selection Operator）回归是一种线性回归的正则化方法，它在损失函数中引入了L1正则化项。LASSO回归具有以下优点和缺点：

优点：

1. 特征选择：LASSO回归具有自动进行特征选择的能力。由于L1正则化项的存在，LASSO可以将某些特征的系数收缩到零，从而实现对无关特征的自动筛选，得到稀疏的模型，简化了模型的复杂度和解释性。
2. 处理高维数据：LASSO回归适用于高维数据集，能够有效应对特征维度远远大于样本数量的情况。通过特征选择，可以降低模型的复杂度和过拟合的风险。
3. 解释性：由于LASSO回归可以将某些特征的系数收缩为零，得到的模型更简洁，更易于解释和理解。

缺点：

1. 参数选择：LASSO回归中的正则化参数（λ）需要事先确定，这需要通过交叉验证等方法来选择最佳的参数值。选择不合适的参数值可能导致欠拟合或过拟合的问题。
2. 稳定性：当数据特征之间存在高度相关性时，LASSO回归倾向于随机选择其中一个相关特征，而将其他相关特征的系数收缩为零。这可能导致模型对于数据的微小变化非常敏感，使模型不够稳定。
3. 无法处理相关特征组：如果数据中存在高度相关的特征组（例如多重共线性），LASSO回归只能选择其中的一个特征，而无法区分它们的重要性。

需要注意的是，LASSO回归在一些特定情况下可能不适用，例如当特征间存在强相关性、特征间的关系是非线性的时候。在实际应用中，需要根据具体问题和数据特点来选择合适的回归方法。

Scikit-learn 内置在 LassoCV 中：

```
from sklearn.linear_model import LassoCV
lasso = LassoCV()
lasso.fit(X, y.ravel())
```

7、岭回归

岭回归（Ridge Regression）是一种线性回归的正则化方法，它在损失函数中引入了L2正则化项。岭回归具有以下优点和缺点：

优点：

1. 缓解多重共线性：岭回归可以有效地应对多重共线性问题，即当自变量之间存在高度相关性时。通过L2正则化项，岭回归可以对高相关性的特征进行惩罚，减小它们的系数，从而降低模型对于特征间关系的过度拟合，提高模型的稳定性和泛化能力。
2. 可解释性：与一般线性回归相比，岭回归可以缩小特征的系数，将对模型的影响限制在合理范围内，更易于解释和理解。这有助于识别对目标变量贡献较大的特征。
3. 鲁棒性：岭回归对于异常值和噪声的影响相对较小，能够提高模型的鲁棒性，降低对异常值的敏感性。

缺点：

1. 特征选择的限制：与LASSO回归相比，岭回归不具备自动进行特征选择的能力。L2正则化项只会使特征的系数趋近于零，而不会将其直接收缩为零。因此，岭回归无法实现特征的稀疏性，无法筛选出无关特征。
2. 参数选择：岭回归中的正则化参数（λ）需要事先确定，这需要通过交叉验证等方法来选择最佳的参数值。选择不合适的参数值可能导致欠拟合或过拟合的问题。
3. 系数偏向：岭回归对于特征系数的估计会有一定的偏向，使得估计值相对于真实值稍微偏大。这是由于L2正则化项的存在，对系数进行了收缩，但没有将系数收缩到零。

Scikit-learn 内置在 RidgeCV 中：

```
from sklearn.linear_model import RidgeCV
ridge = RidgeCV()
ridge.fit(X, y)
```

8、ElasticNet 回归

ElasticNet回归是一种结合了L1和L2正则化的线性回归方法，它在损失函数中同时引入了L1和L2正则化项。ElasticNet回归具有以下优点和缺点：

优点：

1. 特征选择：ElasticNet回归能够同时结合L1和L2正则化的优点，既能够实现特征选择，又能够缓解多重共线性问题。L1正则化项可以将某些特征的系数收缩为零，实现特征的稀疏性和自动筛选；而L2正则化项可以减小特征的系数，降低对特征间相关性的过拟合，提高模型的稳定性。
2. 鲁棒性：ElasticNet回归对于异常值和噪声的影响相对较小，能够提高模型的鲁棒性。
3. 灵活性：ElasticNet回归通过调节L1和L2正则化项之间的权重参数来平衡特征选择和模型稳定性的需求。通过调整正则化参数，可以灵活地控制模型的稀疏性和系数收缩程度。

缺点：

1. 参数选择：ElasticNet回归中有两个正则化参数需要事先确定，即L1正则化项的权重和L2正则化项的权重。选择合适的参数值需要通过交叉验证等方法来进行调优。
2. 计算复杂度：由于同时引入了L1和L2正则化项，ElasticNet回归的计算复杂度相对于岭回归和LASSO回归来说更高一些。
3. 解释性：与岭回归和LASSO回归相比，ElasticNet回归的模型系数更难以解释和理解，因为它同时考虑了两种正则化的影响。

```
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
elastic_net = ElasticNet(alpha=0.5, l1_ratio=0.5)
elastic_net.fit(X_train, y_train) #模型训练
y_pred = elastic_net.predict(X_test) # 模型预测
mse = mean_squared_error(y_test, y_pred) # 模型评估
```

9、xgboost、catboost、LightGBM等等

这些都是非常强的模型，用于分类、预测等，主要就是调参和优化了，需要对机器学习有一定的基础。

深入理解XGBoost

深入理解CatBoost

深入理解LightGBM