**Machine Learning Notes**

This document provides basic knowledge about machine learning, particularly applications to Global Change Ecology Research. I will recursively update this document in order to help myself summarize what I have learned from reading books and articles and from conducting research.

# **Feature importance**

It is noted that not all features are equally important. Some of them are either redundant or unessential. Thus, we need to discard some features in order to

* improve the ability of a trained model
* reduce the over-fitting
* increase the speed of training and deduction /dɪˈdʌkʃ(ə)n/
* enhance the explainability

# **Methods of analyzing feature importance**

## **Permutation importance**

|  |
| --- |
| from sklearn.datasets import load\_breast\_cancer  from sklearn.ensemble import RandomForestClassifier  from sklearn.inspection import permutation\_importance  from sklearn.model\_selection import train\_test\_split  import matplotlib.pyplot as plt  cancer = load\_breast\_cancer()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(cancer.data, cancer.target, random\_state=1)  rf = RandomForestClassifier(n\_estimators=100, random\_state=1)  rf.fit(X\_train, y\_train)  baseline = rf.score(X\_test, y\_test)  result = permutation\_importance(rf, X\_test, y\_test, n\_repeats=10, random\_state=1, scoring='accuracy')  importances = result.importances\_mean  # Visualize permutation importances  plt.bar(range(len(importances)), importances)  plt.xlabel('Feature Index')  plt.ylabel('Permutation Importance')  plt.show()  A graph showing the different types of bars  Description automatically generated with medium confidence |

## **Coef\_feature\_importances**

|  |
| --- |
| def coef\_feature\_importances():  from sklearn.datasets import load\_breast\_cancer  from sklearn.ensemble import RandomForestClassifier  X, y = load\_breast\_cancer(return\_X\_y=True)  rf = RandomForestClassifier(n\_estimators=100, random\_state=1)  rf.fit(X, y)  importances = rf.feature\_importances\_  # Plot importances  plt.bar(range(X.shape[1]), importances)  plt.xlabel('Feature Index')  plt.ylabel('Feature Importance')  plt.savefig('coef\_feature\_importances.jpg')  plt.show() |

## **Leave-one-out**

|  |
| --- |
| from sklearn.datasets import load\_breast\_cancer  from sklearn.model\_selection import train\_test\_split  from sklearn.ensemble import RandomForestClassifier  from sklearn.metrics import accuracy\_score  import matplotlib.pyplot as plt  import numpy as np  # Load sample data  X, y = load\_breast\_cancer(return\_X\_y=True)  # Split data into train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)  # Train a random forest model  rf = RandomForestClassifier(n\_estimators=100, random\_state=1)  rf.fit(X\_train, y\_train)  # Get baseline accuracy on test data  base\_acc = accuracy\_score(y\_test, rf.predict(X\_test))  # Initialize empty list to store importances  importances = []  # Iterate over all columns and remove one at a time  for i in range(X\_train.shape[1]):  X\_temp = np.delete(X\_train, i, axis=1)  rf.fit(X\_temp, y\_train)  acc = accuracy\_score(y\_test, rf.predict(np.delete(X\_test, i, axis=1)))  importances.append(base\_acc - acc)  # Plot importance scores  plt.bar(range(len(importances)), importances)  plt.savefig('leave\_one\_out.jpg')  plt.show()  print("yes") |

## **Correlation analysis**

|  |
| --- |
| def correlation\_analysis():  import pandas as pd  from sklearn.datasets import load\_breast\_cancer  X, y = load\_breast\_cancer(return\_X\_y=True)  df = pd.DataFrame(X, columns=range(30))  df['y'] = y  correlations = df.corrwith(df.y).abs()  correlations.sort\_values(ascending=False, inplace=True)  correlations.plot.bar() |

## **Recursively eliminate features**

|  |
| --- |
| from sklearn.ensemble import RandomForestClassifier  from sklearn.feature\_selection import RFE  import pandas as pd  from sklearn.datasets import load\_breast\_cancer  import matplotlib.pyplot as plt  X, y = load\_breast\_cancer(return\_X\_y=True)  df = pd.DataFrame(X, columns=range(30))  df['y'] = y  rf = RandomForestClassifier()  rfe = RFE(rf, n\_features\_to\_select=10)  rfe.fit(X, y)  print(rfe.ranking\_) |

## **XGBoost**

|  |
| --- |
| import xgboost as xgb  import pandas as pd  from sklearn.datasets import load\_breast\_cancer  import matplotlib.pyplot as plt  X, y = load\_breast\_cancer(return\_X\_y=True)  df = pd.DataFrame(X, columns=range(30))  df['y'] = y  model = xgb.XGBClassifier()  model.fit(X, y)  importances = model.feature\_importances\_  importances = pd.Series(importances, index=range(X.shape[1]))  importances.plot.bar() |

## **PCA**

|  |
| --- |
| from sklearn.decomposition import PCA  import pandas as pd  from sklearn.datasets import load\_breast\_cancer  import matplotlib.pyplot as plt  X, y = load\_breast\_cancer(return\_X\_y=True)  df = pd.DataFrame(X, columns=range(30))  df['y'] = y  pca = PCA()  pca.fit(X)  plt.bar(range(pca.n\_components\_), pca.explained\_variance\_ratio\_)  plt.xlabel('PCA components')  plt.ylabel('Explained Variance') |

## **Variance analysis**

|  |
| --- |
| ## variance analysis  ## the higher a f, the strong importance a feature has ##  from sklearn.feature\_selection import f\_classif  import pandas as pd  from sklearn.datasets import load\_breast\_cancer  import matplotlib.pyplot as plt  X, y = load\_breast\_cancer(return\_X\_y=True)  df = pd.DataFrame(X, columns=range(30))  df['y'] = y  fval = f\_classif(X, y)  fval = pd.Series(fval[0], index=range(X.shape[1]))  fval.plot.bar() |

## **Chi2**

|  |
| --- |
| from sklearn.feature\_selection import chi2  import pandas as pd  from sklearn.datasets import load\_breast\_cancer  import matplotlib.pyplot as plt  X, y = load\_breast\_cancer(return\_X\_y=True)  df = pd.DataFrame(X, columns=range(30))  df['y'] = y  chi\_scores = chi2(X, y)  chi\_scores = pd.Series(chi\_scores[0], index=range(X.shape[1]))  chi\_scores.plot.bar() |

# **Warming Tips**

1. use multiple methods
2. focus on ensemble methods of aggregated results
3. pay more attention to relative order instead of absolute values
4. differences do not mean problems, indicating we should check the reasons behind

# **Cross validation methods**

* K-cross validation
* Leave-one-out validation
* Stratified validation that aims to address the issue of imbalance data

## **Demo**

|  |
| --- |
| library(survival) head(gbsg)  str(gbsg)  *# split it into training and test*  set.seed(123) data <- gbsg[,c(-1)] data$status <- as.factor(data$status) train\_indices <- sample(x = 1:nrow(data), size = 0.85 \* nrow(data), replace = FALSE) test\_indices <- sample(setdiff(1:nrow(data), train\_indices), size = 0.15 \* nrow(data), replace = FALSE) train\_data <- data[train\_indices, ] test\_data <- data[test\_indices, ]  #forward search  #filter features with importance  # the following codes use forward search to determine the optimal set of features  # k-fold  cross validation  k <- 10 fit.control <- trainControl(method = "cv", number = k) fit.rf <- train(status ~ age, data = train\_data, method = "rf", trControl = fit.control) print(fit.rf$results)  # use test data to calculate the accuracy  test.pred <- predict(fit.rf, test\_data, type = "prob") test.pred$prediction <- ifelse(test.pred$"1" >= 0.5, 1, 0) accuracy <- sum(test.pred$prediction == test\_data$status) / length( test\_data$status) accuracy  # age+ meno fit.rf <- train(status ~ age+meno, data = train\_data, method = "rf", trControl = fit.control) print(fit.rf$results)  # use test data to calculate the accuracy test.pred <- predict(fit.rf, test\_data, type = "prob") test.pred$prediction <- ifelse(test.pred$"1" >= 0.5, 1, 0) accuracy <- sum(test.pred$prediction == test\_data$status) / length( test\_data$status) accuracy  > accuracy [1] 0.6078431 > accuracy [1] 0.5588235 |

## **GridSearchCV**

|  |
| --- |
| import xgboost as xgb from sklearn.datasets import load\_boston from sklearn.model\_selection import GridSearchCV  boston = load\_boston() X, y = boston.data, boston.target  xgb\_reg = xgb.XGBRegressor()  param\_grid = {     'min\_child\_weight': [1, 5, 10, 20] }  grid\_search = GridSearchCV(estimator=xgb\_reg, param\_grid=param\_grid, cv=5)  grid\_search.fit(X, y)  print("Best Parameter: ", grid\_search.best\_params\_) print("Best Score: ", grid\_search.best\_score\_) |