

Summer 2023 Data C100 Final Reference Sheet

Pandas

Suppose `df` is a DataFrame; `s` is a Series. `import pandas as pd`

Function	Description
<code>pd.read_csv(filepath, delimiter)</code>	Loads the data file at <code>filepath</code> into a DataFrame with column fields separated by <code>delimiter</code> (',', '\t')
<code>df[col]</code>	Returns the column labeled <code>col</code> from <code>df</code> as a Series.
<code>df[[col1, col2]]</code>	Returns a DataFrame containing the columns labeled <code>col1</code> and <code>col2</code> .
<code>s.loc[rows] / df.loc[rows, cols]</code>	Returns a Series/DataFrame with rows (and columns) selected by their index values.
<code>s.iloc[rows] / df.iloc[rows, cols]</code>	Returns a Series/DataFrame with rows (and columns) selected by their positions.
<code>s.isnull() / df.isnull()</code>	Returns boolean Series/DataFrame identifying missing values
<code>s.fillna(value) / df.fillna(value)</code>	Returns a Series/DataFrame where missing values are replaced by <code>value</code>
<code>s.isin(values) / df.isin(values)</code>	Returns a Series/DataFrame of booleans indicating if each element is in <code>values</code> .
<code>df.drop(labels, axis)</code>	Returns a DataFrame without the rows or columns named <code>labels</code> along <code>axis</code> (either 0 or 1)
<code>df.rename(index=None, columns=None)</code>	Returns a DataFrame with renamed columns from a dictionary <code>index</code> and/or <code>columns</code>
<code>df.sort_values(by, ascending=True)</code>	Returns a DataFrame where rows are sorted by the values in columns <code>by</code>
<code>s.sort_values(ascending=True)</code>	Returns a sorted Series.
<code>s.unique()</code>	Returns a NumPy array of the unique values
<code>s.value_counts()</code>	Returns the number of times each unique value appears in a Series, sorted in descending order of count.
<code>pd.merge(left, right, how='inner', on='a')</code>	Returns a DataFrame joining <code>left</code> and <code>right</code> on the column labeled <code>a</code> ; the join is of type inner
<code>left.merge(right, left_on=col1, right_on=col2)</code>	Returns a DataFrame joining <code>left</code> and <code>right</code> on columns labeled <code>col1</code> and <code>col2</code> .
<code>df.pivot_table(index, columns, values=None, aggfunc='mean')</code>	Returns a DataFrame pivot table where columns are unique values from <code>columns</code> (column name or list), and rows are unique values from <code>index</code> (column name or list); cells are collected <code>values</code> using <code>aggfunc</code> . If <code>values</code> is not provided, cells are collected for each remaining column with multi-level column indexing.
<code>df.set_index(col)</code>	Returns a DataFrame that uses the values in the column labeled <code>col</code> as the row index.
<code>df.reset_index()</code>	Returns a DataFrame that has row index 0, 1, etc., and adds the current index as a column.

Let `grouped = df.groupby(by)` where `by` can be a column label or a list of labels.

Function	Description
<code>grouped.count()</code>	Return a Series containing the size of each group, excluding missing values
<code>grouped.size()</code>	Return a Series containing size of each group, including missing values
<code>grouped.mean()/min()/max()</code>	Return a Series/DataFrame containing mean/min/max of each group for each column, excluding missing values
<code>grouped.filter(f)</code>	Filters or aggregates using the given function <code>f</code>
<code>grouped.agg(f)</code>	

Text Wrangling and Regular Expressions

Pandas str methods

Function	Description
<code>s.str.len()</code>	Returns a Series containing length of each string
<code>s.str[a:b]</code>	Returns a Series where each element is a slice of the corresponding string indexed from a (inclusive, optional) to b (non-inclusive, optional)
<code>s.str.lower()/s.str.upper()</code>	Returns a Series of lowercase/uppercase versions of each string
<code>s.str.replace(pat, repl)</code>	Returns a Series that replaces occurrences of substrings matching the regex pat with string repl
<code>s.str.contains(pat)</code>	Returns a boolean Series indicating if a substring matching the regex pat is contained in each string
<code>s.str.extract(pat)</code>	Returns a Series of the first subsequence of each string that matches the regex pat. If pat contains capturing group(s), outputs a DataFrame with one column for each group.
<code>s.str.split(pat)</code>	Splits the strings in s at the delimiter pat. Returns a Series of lists, where each list contains strings of the characters before and after the split.

Regex patterns

Operator	Description	Operator	Description
.	Matches any character except \n	*	Matches preceding character/group zero or more times
\	Escapes metacharacters	+	Matches preceding character/group one or more times
	Matches expression on either side of expression; has lowest priority of any operator	^	Matches the beginning of the string
\d, \w, \s	Predefined character group of digits (0-9), alphanumerics (a-z, A-Z, 0-9, and underscore), or whitespace, respectively	\$	Matches the end of the string
\D, \W, \S	Inverse sets of \d, \w, \s, respectively	()	Capturing group or sub-expression
{m}	Matches preceding character/group exactly m times	[]	Character class used to match any of the specified characters or range (e.g. [abcde] is equivalent to [a-e])
{m, n}	Matches preceding character/group at least m times and at most n times. If either m or n are omitted, set lower/upper bounds to 0 and infinity, respectively	[^]	Invert character class; e.g. [^a-c] matches all characters except a, b, c

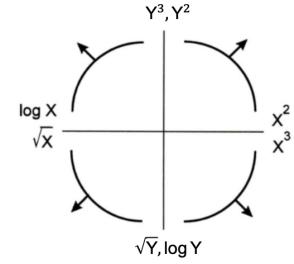
Python re methods

Function	Description
<code>re.match(pattern, string)</code>	Returns all matching characters if zero or more characters at beginning of string matches pattern, else None
<code>re.search(pattern, string)</code>	Returns all matching characters if zero or more characters anywhere in string matches pattern, else None
<code>re.findall(pattern, string)</code>	Returns a list of all non-overlapping matches of pattern in string (if none, returns empty list). If pattern includes capturing groups, only return captured characters.
<code>re.sub(pattern, repl, string)</code>	Returns string after replacing all occurrences of pattern with repl

Visualization

Matplotlib: x and y are sequences of values. `import matplotlib.pyplot as plt`

Tukey-Mosteller Bulge Diagram.



Seaborn: x and y are keyword arguments assigned to string column names in a DataFrame data. `import seaborn as sns`

Function	Description
<code>sns.countplot(data=None, x=None)</code>	Create a barplot of value counts of variable x from data
<code>sns.histplot(data=None, x=None, stat='count', kde=False)</code>	Creates a histogram of x from data, where bin statistics stat is one of 'count', 'frequency', 'probability', 'percent', and 'density'; optionally overlay a kernel density estimator.
<code>sns.displot(data=None, x=None, stat='count', rug=False, kde=True)</code>	
<code>sns.boxplot(data=None, x=None, y=None)</code>	Create a boxplot of a numeric feature (e.g., y), optionally factoring by a category (e.g., x), from data. violinplot is similar but also draws a kernel density estimator of the numeric feature
<code>sns.violinplot(data=None, x=None, y=None)</code>	
<code>sns.scatterplot(data=None, x=None, y=None)</code>	Create a scatterplot of x versus y from data
<code>sns.lmplot(data=None, x=None, y=None, fit_reg=True)</code>	Create a scatterplot of x versus y from data, and by default overlay a least-squares regression line
<code>sns.jointplot(data=None, x=None, y=None, kind)</code>	Combine a bivariate scatterplot of x versus y from data, with univariate density plots of each variable overlaid on the axes; kind determines the visualization type for the distribution plot, can be scatter, kde or hist
<code>sns.kdeplot(data=None, x=None)</code>	Create a kernel density estimate (KDE) of the distribution of x from data

Modeling

Concept	Formula	Concept	Formula
Variance, σ_x^2	$\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$	Correlation r	$r = \frac{1}{n} \sum_{i=1}^n \frac{x_i - \bar{x}}{\sigma_x} \frac{y_i - \bar{y}}{\sigma_y}$
L_1 loss	$L_1(y, \hat{y}) = y - \hat{y} $	Linear regression estimate of y	$\hat{y} = \theta_0 + \theta_1 x$
L_2 loss	$L_2(y, \hat{y}) = (y - \hat{y})^2$	Least squares linear regression	$\hat{\theta}_0 = \bar{y} - \hat{\theta}_1 \bar{x} \quad \hat{\theta}_1 = r \frac{\sigma_y}{\sigma_x}$
Empirical risk with loss L	$R(\theta) = \frac{1}{n} \sum_{i=1}^n L(y_i, \hat{y}_i)$		

Ordinary Least Squares

Multiple Linear Regression Model: $\hat{Y} = \mathbb{X}\theta$ with design matrix \mathbb{X} , response vector \mathbb{Y} , and predicted vector \hat{Y} . If there are p features plus a bias/intercept, then the vector of parameters $\theta = [\theta_0, \theta_1, \dots, \theta_p]^T \in \mathbb{R}^{p+1}$. The vector of estimates $\hat{\theta}$ is obtained from fitting the model to the sample (\mathbb{X}, \mathbb{Y}) .

Concept	Formula	Concept	Formula
Mean squared error	$R(\theta) = \frac{1}{n} \ \mathbb{Y} - \mathbb{X}\theta\ _2^2$	Normal equation	$\mathbb{X}^T \mathbb{X} \hat{\theta} = \mathbb{X}^T \mathbb{Y}$
Least squares estimate, if \mathbb{X} is full rank	$\hat{\theta} = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T \mathbb{Y}$	Residual vector, e	$e = \mathbb{Y} - \hat{Y}$
		Multiple R^2 (coefficient of determination)	$R^2 = \frac{\text{variance of fitted values}}{\text{variance of } y}$

Regularization

Concept	Formula	Concept	Formula
Ridge Regression L2 Regularization	$\frac{1}{n} \ \mathbb{Y} - \mathbb{X}\theta\ _2^2 + \lambda \ \theta\ _2^2$	Squared L2 Norm of $\theta \in \mathbb{R}^p$	$\ \theta\ _2^2 = \sum_{j=1}^p \theta_j^2$
Ridge regression estimate (closed form)	$\hat{\theta}_{\text{ridge}} = (\mathbb{X}^T \mathbb{X} + n\lambda I)^{-1} \mathbb{X}^T \mathbb{Y}$		
LASSO Regression L1 Regularization	$\frac{1}{n} \ \mathbb{Y} - \mathbb{X}\theta\ _2^2 + \lambda \ \theta\ _1$	L1 Norm of $\theta \in \mathbb{R}^p$	$\ \theta\ _1 = \sum_{j=1}^p \theta_j $

Gradient Descent

Let L be an objective function to minimize with respect to θ ; assume that some optimal parameter vector $\hat{\theta}$ exists. Suppose $\theta^{(0)}$ is some starting estimate at $t = 0$, and $\theta^{(t)}$ is the estimate at step t . Then for a learning rate α , the gradient update step to compute $\theta^{(t+1)}$ is:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_{\theta} L$$

where $\nabla_{\theta} L$ is the partial derivative/gradient of L with respect to θ , evaluated at $\theta^{(t)}$.

Classification and Logistic Regression

Confusion Matrix

Columns are the predicted values \hat{y} and rows are the actual classes y .

$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	True negative (TN) False Positive (FP)
$y = 1$	False negative (FN) True Positive (TP)

Classification Performance

Suppose you predict n datapoints.

Metric	Formula	Other Names
Accuracy	$\frac{TP+TN}{n}$	
Precision	$\frac{TP}{TP+FP}$	
Recall/TPR	$\frac{TP}{TP+FN}$	True Positive Rate, Sensitivity
FPR	$\frac{FP}{FP+TN}$	False Positive Rate, Specificity

An ROC curve visualizes TPR vs. FPR for different thresholds T .

Logistic Regression Model: For input feature vector x , $\hat{P}_{\theta}(Y = 1|x) = \sigma(x^T \theta)$, where $\sigma(z) = 1/(1 + e^{-z})$. The estimate $\hat{\theta}$ is the parameter θ that minimizes the average cross-entropy loss on training data. For a single datapoint, define cross-entropy loss as $-\log(p) + (1 - p) \log(1 - p)$, where p is the probability that the response is 1.

Logistic Regression Classifier: For a given input x and trained logistic regression model with parameter θ , compute $p = \hat{P}(Y = 1|x) = \sigma(x^T \theta)$. predict response \hat{y} with classification threshold T as follows:

$$\hat{y} = \text{classify}(x) = \begin{cases} 1 & p \geq T \\ 0 & \text{otherwise} \end{cases}$$

Scikit-Learn

Package: `sklearn.linear_model`

Linear	Logistic	Class/Function(s)	Description
✓	-	<code>LinearRegression(fit_intercept=True)</code>	Returns an ordinary least squares Linear Regression model.
-	✓	<code>LogisticRegression(fit_intercept=True, penalty='l2', C=1.0)</code>	Returns an ordinary least squares Linear Regression model. Hyperparameter C is inverse of regularization parameter, $C = 1/\lambda$.
✓	-	<code>Lasso()</code> , <code>Ridge()</code>	Returns a Lasso (L1 Regularization) or Ridge (L2 regularization) linear model, respectively.
✓	✓	<code>model.fit(X, y)</code>	Fits the scikit-learn <code>model</code> to the provided X and y.
✓	✓	<code>model.predict(X)</code>	Returns predictions for the X passed in according to the fitted <code>model</code> .
-	✓	<code>model.predict_proba(X)</code>	Returns predicted probabilities for X passed in according to the fitted <code>model</code> . If binary classes, returns probabilities for both classes 0 and 1.
✓	✓	<code>model.coef_</code>	Estimated coefficients for the linear model, not including the intercept.
✓	✓	<code>model.intercept_</code>	Bias/intercept term of the linear model. Set to 0.0 if <code>fit_intercept=False</code> .

Package: `sklearn.model_selection`

Function	Description
<code>train_test_split(*arrays, test_size=0.2)</code>	Returns two random subsets of each array passed in, with 0.8 of the array in the first subset and 0.2 in the second subset.

Probability

Let X have a discrete probability distribution $P(X = x)$. X has expectation $\mathbb{E}[X] = \sum_x xP(X = x)$ over all possible values x , variance $\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$, and standard deviation $\text{SD}(X) = \sqrt{\text{Var}(X)}$.

The covariance of two random variables X and Y is $\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$. If X and Y are independent, then $\text{Cov}(X, Y) = 0$.

Notes	Property of Expectation	Property of Variance
X is a random variable.		$\text{Var}(X) = E[X^2] - (E[X])^2$
X is a random variable, $a, b \in \mathbb{R}$ are scalars.	$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b$	$\text{Var}(aX + b) = a^2\text{Var}(X)$
X, Y are random variables.	$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$	$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$
X is a Bernoulli random variable that takes on value 1 with probability p and 0 otherwise.	$\mathbb{E}[X] = p$	$\text{Var}(X) = p(1 - p)$

Central Limit Theorem

Let (X_1, \dots, X_n) be a sample of independent and identically distributed random variables drawn from a population with mean μ and standard deviation σ . The sample mean $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$ is normally distributed, where $\mathbb{E}[\bar{X}_n] = \mu$ and $\text{SD}(\bar{X}_n) = \sigma/\sqrt{n}$.

Model Risk

Suppose for each individual with fixed input x , we observe a random response $Y = g(x) + \epsilon$, where g is the true relationship and ϵ is random noise with zero mean and variance σ^2 .

For a new individual with fixed input x , define our random prediction $\hat{Y}(x)$ based on a model fit to our observed sample (\mathbb{X}, \mathbb{Y}) . The model risk is the mean squared prediction error between Y and $\hat{Y}(x)$: $\mathbb{E}[(Y - \hat{Y}(x))^2] = \sigma^2 + (\mathbb{E}[\hat{Y}(x)] - g(x))^2 + \text{Var}(\hat{Y}(x))$.

SQL

```
SELECT [DISTINCT]
    {* | expr [[AS] c_alias]
    ,expr [[AS] c_alias] ...}
FROM tableref {, tableref}
[[INNER | LEFT ] JOIN table_name ON qualification_list]
[WHERE search_condition]
[GROUP BY colname {,colname...}]
[HAVING search_condition]
[ORDER BY column_list]
[LIMIT number]
[OFFSET number of rows];
```

Syntax	Description
SELECT column_expression_list	List is comma-separated. Column expressions may include aggregation functions (MAX, SUM, COUNT, AVG, etc). AS renames columns. DISTINCT selects only unique rows.
WHERE a IN cons_list	Select rows for which the value in column a is among the values in a cons_list.
ORDER BY RANDOM() LIMIT n	Draw a simple random sample of n rows.
ORDER BY a, b DESC	Order by column a (ascending by default) , then b (descending).
CASE WHEN pred THEN cons ELSE alt END	Evaluates to cons if pred is true and alt otherwise. Multiple WHEN/THEN pairs can be included, and ELSE is optional.
WHERE s.a LIKE 'p'	Matches each entry in the column a of table s to the text pattern p. The wildcard % matches at least zero characters. _ matches exactly one character.
LIMIT number	Keep only the first number rows in the return result.
OFFSET number	Skip the first number rows in the return result.

Principal Component Analysis (PCA)

The i -th Principal Component of the matrix X is defined as the i -th column of $U\Sigma$ defined by Singular Value Decomposition (SVD).

$X = U\Sigma V^T$ is the SVD of X if U and V^T are matrices with orthonormal columns and Σ is a diagonal matrix. The diagonal entries of Σ , $[s_1, \dots, s_r, 0, \dots, 0]$, are known as singular values of X , where $s_i > s_j$ for $i < j$ and $r = \text{rank}(X)$.

Define the design matrix $X \in \mathbb{R}^{n \times p}$. Define the total variance of X as the sum of individual variances of the p features. The amount of variance captured by the i -th principal component is equivalent to s_i^2/n , where n is the number of datapoints.

Syntax	Description
np.linalg.svd(X, full_matrices = True)	SVD of X with shape (M, N) that returns u, s, vt, where s is a 1D array of X's singular values. If full_matrices=True, u and vt have shapes (M, M) and (N, N) respectively; otherwise shapes are (M, K) and (K, N), respectively, where K = min(M, N).

Decision Trees

Suppose you have a **decision tree** classifier for k classes. For each node, define the probability for class $C \in \{1, \dots, k\}$ as $p_C = d_C/d$, where d_C is the number of datapoints in class C (of the total d in the node). The entropy of the node (in bits) is defined as $S = -\sum_C p_C \log_2 p_C$, and the weighted entropy of the split is the average entropy of child nodes weighted by the number of datapoints in each.

Decision tree generation algorithm: all of the data starts in the root node. Repeat until every node is either pure or unsplittable.

- Pick the best feature x and best split value β to maximize the change in weighted entropy.
- Split data into two nodes, one where $x < \beta$, and one where $x \geq \beta$

A node that only has samples from one class is called a "pure" node. A node that has overlapping datapoints from different classes and thus cannot be split is called "unsplittable."

A **random forest** is a collection of many decision trees fit to variations of the same training data (e.g. bootstrapped samples, also called bagging). It is an ensemble method.