Project 3: Concrete Strength Prediction

This manuscript (permalink) was automatically generated from qinyuz2/project3@cba62a6 on December 6, 2020.

Authors

- **Qinyu Zhang**
  · qinyuz2
  Department of Civil & Environmental Engineering, University of Illinois at Urbana-Champaign

- **Jane Roe**
  XXXX-XXXX-XXXX-XXXX · janeroe
  Department of Something, University of Whatever; Department of Whatever, University of Something

Introduction

The second most consumed substance in the world after water is concrete. Currently, the world produces 4.4 billion of concrete annually. Conventional concrete is a mixture of cement, aggregates (coarse and fine) and water. Many admixtures like mineral and chemical are incorporated in concrete to improve its performance. The main idea is not only to improve its overall performance and durability but also to reduce the emission of carbon dioxide produced by concrete industry itself. Cement production itself is highly energy intensive process. In 2015, it generated around 2.8 billion of CO2 (8% of total). Lots of research have been carried out to decrease the percentage of cement in concrete by using different substitutes. Researchers are opting for more environment friendly and sustainable options. This new idea of unconventional concrete requires backup and standards. The conventional method of testing concrete's strength is to cast concrete cylinders or cubes with different mix ratio of its constituents and test them after 7, 14 or 28 days. This method requires significant amount of labor and time.

2. Method

3. Discussion

3.1 Model Training and Evaluation

3.1.1 Linear Regression

Linear Regression is the simplest but powerful model. In the previous studies, it was widely used in the prediction of concrete strength. This model assumes a linear relationship between independent and dependent variables.

Code for Linear Regression Model

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(train_x, train_y)
y_pred_lin= lin_reg.predict(test_x)
```

Accuracy of Model is: 0.5177053629131334

Root Mean Squared Error of Model is: 11.420285520195613

3.1.2 Lasso Method

The Lasso is a shrinkage and selection method for linear regression. It minimizes the usual sum of squared errors, with a bound on the sum of the absolute values of the coefficients.

Code for Lasso Model

```
from sklearn.linear_model import Lasso

las = Lasso(alpha=0.1)
model2 = las.fit(train_x, train_y)
predictions2 = las.predict(test_x)
```

Accuracy of Model is: 0.38967572787640603

Root Mean Squared Error of Model is: 11.988560504390488

3.1.3 K-nearest Neighbor

The k-nearest neighbor is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems.

Code for K-nearest Neighbor Model

```
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor()
model3=knn.fit(train_x, train_y)
predictions3 = knn.predict(test_x)
```

Accuracy of Model is: 0.34345271643724284

Root Mean Squared Error of Model is: 12.434253663809674

3.1.4 Support Vector Machine

Support vector machine is a supervised machine learning algorithm used for classification, regression and outlier detection. We use a linear Support Vector Machine model.

Code for Support Vector Machine Model

```
from sklearn.svm import SVR

svm= SVR(kernel='linear')
model4=svm.fit(train_x, train_y)
predictions4 = svm.predict(test_x)
```

Accuracy of Model is: 0.34808748601553163

Root Mean Squared Error of Model is: 12.390287319386937

3.1.5 Neural Network

Neural Nework is also used in previous studies to predict concrete strength. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data. A neural network consists of input layer, hidden layers and output layer. It's able to learn and model non-linear and complex relationships between independent and dependent variables.

Code for Neural Network Model

```
import tensorflow as tf

layer_width = 128
l1 = 0.0
l2 = 0.05

model1_split = tf.keras.Sequential()

model1_split.add(tf.keras.layers.Dense(512, activation="relu",
        kernel_regularizer = tf.keras.regularizers.l1_l2(l1=l1, l2=l2)))
model1_split.add(tf.keras.layers.BatchNormalization())
model1_split.add(tf.keras.layers.Dense(256, activation="relu"))
model1_split.add(tf.keras.layers.BatchNormalization())
model1_split.add(tf.keras.layers.Dense(128, activation="relu"))
model1_split.add(tf.keras.layers.BatchNormalization())
model1_split.add(tf.keras.layers.Dense(128, activation="relu"))

model1_split.add(tf.keras.layers.Dense(32, activation="relu"))

model1_split.add(tf.keras.layers.Dense(1))

model1_split.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0015),
                 loss='mean_squared_error',
                 metrics=[tf.keras.metrics.RootMeanSquaredError()]
                 )

history_split = model1_split.fit(train_x, train_y, batch_size=1000,
                  epochs=500, shuffle=True)
epochs_split = history_split.epoch
hist_split = pd.DataFrame(history_split.history)
rmse_split = hist_split["root_mean_squared_error"]

model1_split.summary()

model1_split_prediction = model1_split.predict(test_x)
```

Accuracy of Model is: 0.8565038038524211

Root Mean Squared Error of Model is: 6.229324285295386

3.1.6 Decision Tree Regressor

Decision-tree algorithm is a kind of supervised learning algorithms. It can be used in classification and regression problems.

Code for Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

dtregressor = DecisionTreeRegressor(random_state = 0, min_samples_split=5)
dtregressor.fit(train_x, train_y)
y_pred_dt= dtregressor.predict(test_x)
```

Accuracy of Model is: 0.8204795438030693

Root Mean Squared Error of Model is: 6.9675118677407

3.1.7 Random Forest Regression

Random Forest Regression is a type of supervised learning algorithms. It constructs multiple decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the prediction accuracy and it controls over-fitting as well.

Code for Hyperparameter Tuning

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor

#Use the random grid to search for best hyperparameter
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num =
        10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

# First create the base model to tune
rf_split = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random_split = RandomizedSearchCV(estimator = rf_split,
        param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2,
        random_state=None, n_jobs = -1)
# Fit the random search model
rf_random_split.fit(train_x, train_y)
```

The best parameters for the model is:

{‘n_estimators’: 1200, ‘min_samples_split’: 2, ‘min_samples_leaf’: 1, ‘max_features’: ‘auto’, ‘max_depth’: 100, ‘bootstrap’: True}

Code for Random Forest Model

```
# create a model
model2_split = RandomForestRegressor(n_estimators =
        rf_random_split.best_params_['n_estimators'],
                              min_samples_split =
        rf_random_split.best_params_['min_samples_split'],
                              min_samples_leaf =
        rf_random_split.best_params_['min_samples_leaf'],
                              max_features =
        rf_random_split.best_params_['max_features'],
                              max_depth =
        rf_random_split.best_params_['max_depth'],
                              bootstrap =
        rf_random_split.best_params_['bootstrap'])

model2_split.fit(train_x, train_y)

model2_split_prediction = model2_split.predict(test_x)
```

Accuracy of Model is: 0.9053846430868259

Root Mean Squared Error of Model is: 5.058264806966998

3.2 Comparison and Result

The accuracy and root mean squared error are two parameters to evaluate the model performance. The two parameters of all the seven models we used are shown in the following table.

Table: Summary of Model Performance

Method	**Accuracy**	**Root Mean Squared Error**
Linear Regression	0.5177053629131334	11.420285520195613
Lasso Method	0.38967572787640603	11.988560504390488
K-nearest Neighbor	0.34345271643724284	12.434253663809674
Support Vector Machine	0.34808748601553163	12.390287319386937
Neural Network	0.8565038038524211	6.229324285295386
Decision Tree Regressor	0.8204795438030693	6.9675118677407
Random Forest Regression	0.9053846430868259	5.058264806966998

As shown in the table, Linear Regression, Lasso Method, K-nearest Neighbor and Support Vector Machine do not perform well in predicting concrete strength. The other three methods: Neural Network, Decision Tree Regressor and Random Forest Regression are relatively better.

The accuracy of Random Forest Regression and Neural Network are in the first and second place, respectively. The comparison of these two methods is as follows.

- When creating a model, Neural Network is more complicated. Setting appropriate values for its parameter such as layer numbers, learning rate, batch size, etc. is extremely improtant to the performace of the model, so that it requires more efforts to find better parameters. Random Forest is much easier to find the best parameters.
- Random Forest is less computationally expensive. It can be trained faster than Neural Network.
- When using Neural Network, we should pay attention to avoiding overfitting. However, Random Forest is less prone to overfitting.

Compared with previous studies, the linear regression and neural network models we used do not show the same level of accuracy. One reason could be that the dataset we used to train the model is not big enough. Since we split 20% of the train dataset into test dataset for evaluating model performance, the total number of data we used for training is 565 rows. In the future study, we are expected to use larger dataset to train models, hoping to get a similar accuracy.

What's more, a key insight of our project is that random forest regression shows a great potential to predict concrete strength, however, it has not been used widely in previous studies. We will further confirm wether random forest regression only shows a great performance in the dataset we used or can be applied to other dataset.

For this project, Random Forest Model works best. We recreate a random forest model using the whole given train dataset and use it to predict the given test dataset.

Code for Hyperparameter Tuning

```
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
        random_grid, n_iter = 100, cv = 3, verbose=2, random_state=None,
        n_jobs = -1)
# Fit the random search model
rf_random.fit(x, y)
```

The best parameters for the model is:

{'n_estimators': 600, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 70, 'bootstrap': True}

Code for recreate a model

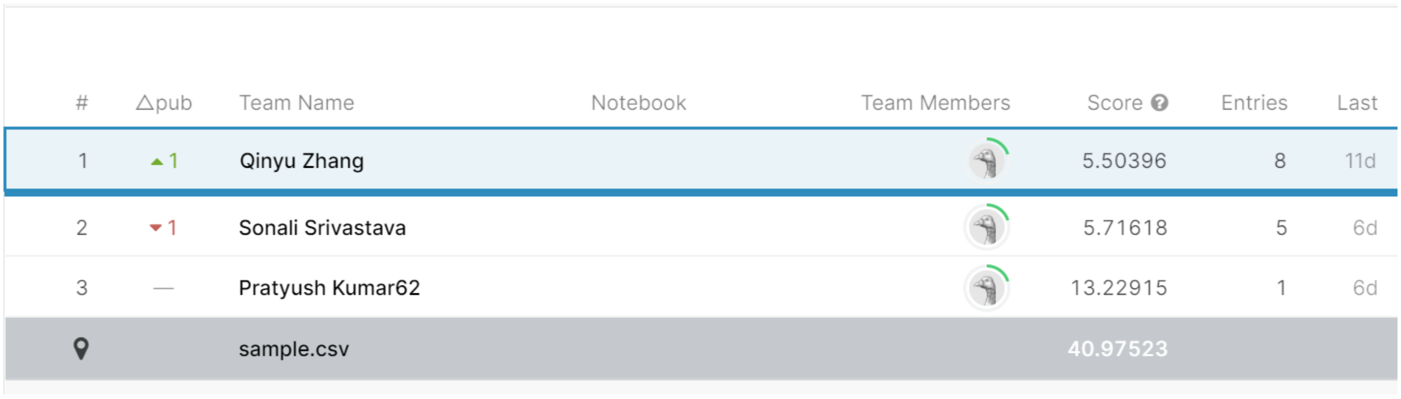
```
model2 = RandomForestRegressor(n_estimators =
        rf_random.best_params_['n_estimators'],
                              min_samples_split =
        rf_random.best_params_['min_samples_split'],
                              min_samples_leaf =
        rf_random.best_params_['min_samples_leaf'],
                              max_features =
        rf_random.best_params_['max_features'],
                              max_depth =
        rf_random.best_params_['max_depth'],
                              bootstrap =
        rf_random.best_params_['bootstrap'])

model2.fit(x, y)
```

prediction Code for recreate a model

```
prediction2 = model2.predict(test_df)
prediction2 = pd.DataFrame(prediction2, columns=['Concrete compressive
        strength(MPa, megapascals)'])
prediction2.index.name='index'
prediction2.to_csv('prediction2.csv')
```

After submitting the *prediction2.csv* to Kaggle Competition, it provides a score of 5.50396 which indicates Root Mean Squared Error. We can expect a model accuracy of 85%-90% for such a score.



#	△pub	Team Name	Notebook	Team Members	Score	Entries	Last
1	▲1	Qinyu Zhang			5.50396	8	11d
2	▼1	Sonali Srivastava			5.71618	5	6d
3	—	Pratyush Kumar62			13.22915	1	6d
		sample.csv			40.97523		

Figure: Kaggle Competition Page

4. Conclusion

This manuscript is a template (aka "rootstock") for Manubot, a tool for writing scholarly manuscripts. Use this template as a starting point for your manuscript.

The rest of this document is a full list of formatting elements/features supported by Manubot. Compare the input (`.md` files in the `/content` directory) to the output you see below.

Basic formatting

Bold text

Semi-bold text

Centered text

Right-aligned text

Italic text

Combined *italics and **bold***

~~Strikethrough~~

1. Ordered list item
2. Ordered list item
    a. Sub-item
    b. Sub-item
        i. Sub-sub-item
3. Ordered list item
    a. Sub-item

- List item
- List item
- List item

subscript: H_2O is a liquid

superscript: 2^{10} is 1024.

unicode superscripts^{0123456789}

unicode subscripts$_{0123456789}$

A long paragraph of text. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Putting each sentence on its own line has numerous benefits with regard to editing and version control.

Line break without starting a new paragraph by putting
two spaces at end of line.

Document organization

Document section headings:

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

A heading centered on its own printed page

Horizontal rule:

`Heading 1`'s are recommended to be reserved for the title of the manuscript.

`Heading 2`'s are recommended for broad sections such as *Abstract*, *Methods*, *Conclusion*, etc.

`Heading 3`'s and `Heading 4`'s are recommended for sub-sections.

Links

Bare URL link: https://manubot.org

Long link with lots of words and stuff and junk and bleep and blah and stuff and other stuff and more stuff yeah

Link with text

Link with hover text

Link by reference

Citations

Citation by DOI [1].

Citation by PubMed Central ID [2].

Citation by PubMed ID [3].

Citation by Wikidata ID [4].

Citation by ISBN [5].

Citation by URL [6].

Citation by alias [7].

Multiple citations can be put inside the same set of brackets [1,5,7]. Manubot plugins provide easier, more convenient visualization of and navigation between citations [2,3,7,8].

Citation tags (i.e. aliases) can be defined in their own paragraphs using Markdown's reference link syntax:

Referencing figures, tables, equations

Figure 1

Figure 2

Figure 3

Figure 4

Table 1

Equation 1

Equation 2

Quotes and code

> Quoted text

> Quoted block of text
>
> Two roads diverged in a wood, and I—
> I took the one less traveled by,
> And that has made all the difference.

Code `in the middle` of normal text, aka `inline code`.

Code block with Python syntax highlighting:

```python
from manubot.cite.doi import expand_short_doi

def test_expand_short_doi():
    doi = expand_short_doi("10/c3bp")
    # a string too long to fit within page:
    assert doi == "10.25313/2524-2695-2018-3-vliyanie-enhansera-copia-i-
        insulyatora-gypsy-na-sintez-ernk-modifikatsii-hromatina-i-
        svyazyvanie-insulyatornyh-belkov-vtransfetsirovannyh-geneticheskih-
        konstruktsiyah"
```

Code block with no syntax highlighting:

```
Exporting HTML manuscript
Exporting DOCX manuscript
Exporting PDF manuscript
```

Figures

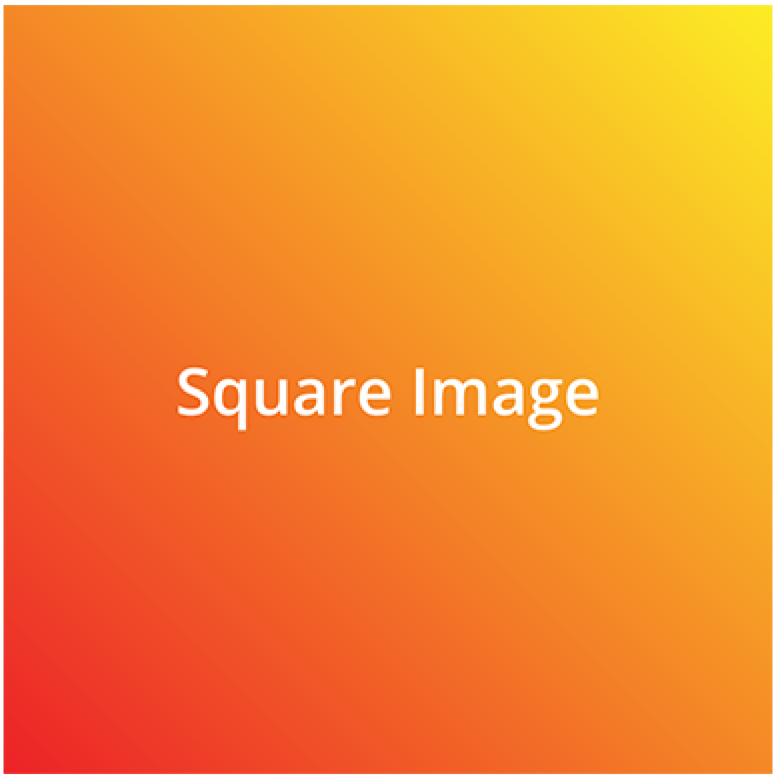


Figure 1: A square image at actual size and with a bottom caption. Loaded from the latest version of image on GitHub.

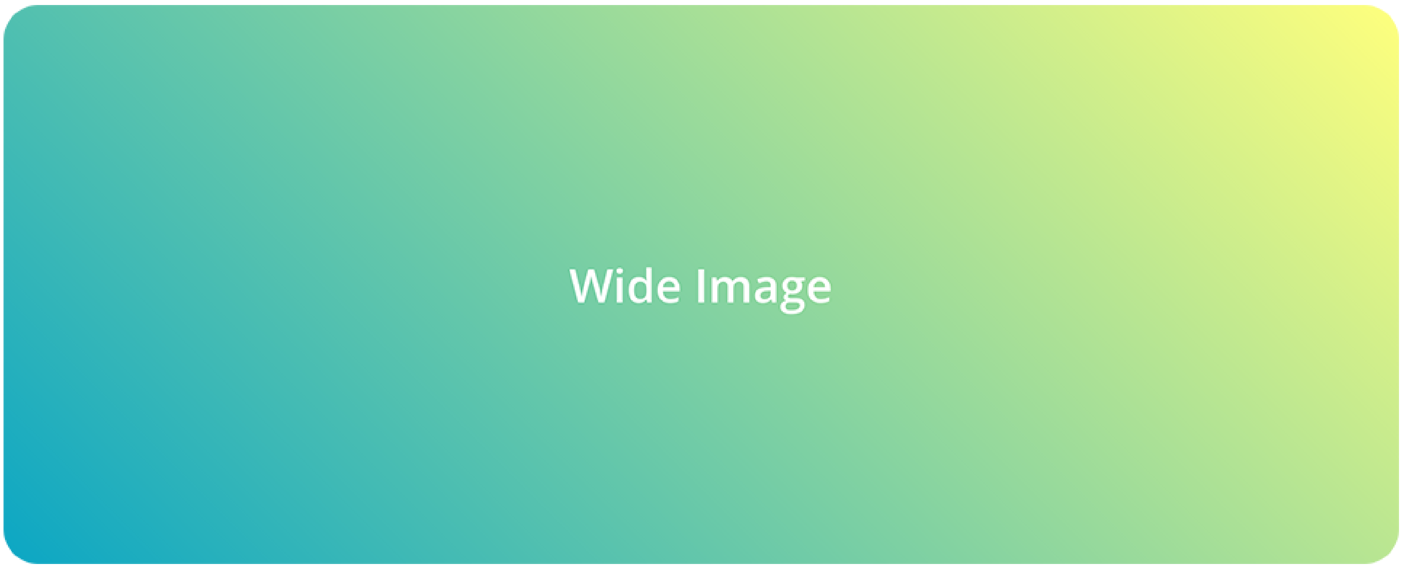


Figure 2: An image too wide to fit within page at full size. Loaded from a specific (hashed) version of the image on GitHub.

Figure 3: A tall image with a specified height. Loaded from a specific (hashed) version of the image on GitHub.

Figure 4: A vector `.svg` image loaded from GitHub. The parameter `sanitize=true` is necessary to properly load SVGs hosted via GitHub URLs. White background specified to serve as a backdrop for transparent sections of the image.

Tables

Table 1: A table with a top caption and specified relative column widths.

Bowling Scores	**Jane**	**John**	**Alice**	**Bob**
Game 1	150	187	210	105
Game 2	98	202	197	102
Game 3	123	180	238	134

Table 2: A table too wide to fit within page.

	Digits 1-33	**Digits 34-66**	**Digits 67-99**	**Ref.**
pi	3.14159265358979323 846264338327950	28841971693993751 0582097494459230	78164062862089986 2803482534211706	`piday.org`
e	2.71828182845904523 536028747135266	24977572470936999 5957496696762772	40766303535475945 7138217852516642	`nasa.gov`

Table 3: A table with merged cells using the `attributes` plugin.

	Colors	
Size	**Text Color**	**Background Color**
big	blue	orange
small	black	white

Equations

A LaTeX equation:

$$\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2} \tag{1}$$

An equation too long to fit within page:

$$\begin{aligned} x = a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + r + s + t \\ + u + v + w + x + y + z + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 \end{aligned} \tag{2}$$

Special

⚠ WARNING *The following features are only supported and intended for `.html` and `.pdf` exports. Journals are not likely to support them, and they may not display correctly when converted to other formats such as `.docx`.*

LINK STYLED AS A BUTTON

Adding arbitrary HTML attributes to an element using Pandoc's attribute syntax:

> Manubot Manubot Manubot Manubot Manubot. Manubot Manubot Manubot Manubot. Manubot Manubot Manubot. Manubot Manubot. Manubot.

Adding arbitrary HTML attributes to an element with the Manubot `attributes` plugin (more flexible than Pandoc's method in terms of which elements you can add attributes to):

> Manubot Manubot Manubot Manubot Manubot. Manubot Manubot Manubot Manubot. Manubot Manubot Manubot. Manubot Manubot. Manubot.

Available background colors for text, images, code, banners, etc:

`white` `lightgrey` `grey` `darkgrey` `black` `lightred` `lightyellow` `lightgreen` `lightblue` `lightpurple` `red` `orange` `yellow` `green` `blue` `purple`

Using the Font Awesome icon set:

✔ ? ★ 🔔 ⊗ ⋯

Light Grey Banner
useful for *general information* - manubot.org

Blue Banner
useful for *important information* - manubot.org

Light Red Banner
useful for *warnings* - manubot.org

References

1. **Sci-Hub provides access to nearly all scholarly literature**
   Daniel S Himmelstein, Ariel Rodriguez Romero, Jacob G Levernier, Thomas Anthony Munro, Stephen Reid McLaughlin, Bastian Greshake Tzovaras, Casey S Greene
   eLife (2018-03-01) https://doi.org/ckcj
   DOI: 10.7554/elife.32822 · PMID: 29424689 · PMCID: PMC5832410

2. **Reproducibility of computational workflows is automated using continuous analysis**
   Brett K Beaulieu-Jones, Casey S Greene
   Nature biotechnology (2017-04) https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6103790/
   DOI: 10.1038/nbt.3780 · PMID: 28288103 · PMCID: PMC6103790

3. **Bitcoin for the biological literature.**
   Douglas Heaven
   Nature (2019-02) https://www.ncbi.nlm.nih.gov/pubmed/30718888
   DOI: 10.1038/d41586-019-00447-9 · PMID: 30718888

4. **Plan S: Accelerating the transition to full and immediate Open Access to scientific publications**
   cOAlition S
   (2018-09-04) https://www.wikidata.org/wiki/Q56458321

5. **Open access**
   Peter Suber
   MIT Press (2012)
   ISBN: 9780262517638

6. **Open collaborative writing with Manubot**
   Daniel S. Himmelstein, Vincent Rubinetti, David R. Slochower, Dongbo Hu, Venkat S. Malladi, Casey S. Greene, Anthony Gitter
   Manubot (2020-05-25) https://greenelab.github.io/meta-review/

7. **Opportunities and obstacles for deep learning in biology and medicine**
   Travers Ching, Daniel S. Himmelstein, Brett K. Beaulieu-Jones, Alexandr A. Kalinin, Brian T. Do, Gregory P. Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M. Hoffman, … Casey S. Greene
   Journal of The Royal Society Interface (2018-04-04) https://doi.org/gddkhn
   DOI: 10.1098/rsif.2017.0387 · PMID: 29618526 · PMCID: PMC5938574

8. **Open collaborative writing with Manubot**
   Daniel S. Himmelstein, Vincent Rubinetti, David R. Slochower, Dongbo Hu, Venkat S. Malladi, Casey S. Greene, Anthony Gitter
   PLOS Computational Biology (2019-06-24) https://doi.org/c7np
   DOI: 10.1371/journal.pcbi.1007128 · PMID: 31233491 · PMCID: PMC6611653