Assignment #13 - Protein function annotation

In this assignment, you will use machine learning techniques to predict the functions of proteins based on their sequences. You will use Python or R to fit a model to a dataset of protein sequences and their known functions, make predictions on a test set, and evaluate the performance of the model.

Requirements

- Python or R
- `scikit-learn` (Python) or `caret` (R) library for machine learning
- `matplotlib` (Python) or `ggplot2` (R) library for visualization

Step 1: Install the necessary libraries

Use the package manager to install the `scikit-learn` library for Python or the `caret` package for R.

Python

```
pip install scikit-learn
```

R

```
install.packages("caret")
```

1. Use machine learning algorithms to predict the functions of the proteins. You can use functions such as `fit()` and `predict()` in Python or `train()` and `predict()` in R to fit a model to the training data and make predictions on the testing data.
2. Evaluate the performance of the model using evaluation metrics such as accuracy, precision, and recall. You can use functions such as `accuracy_score()` in Python or `confusionMatrix()` in R to calculate these metrics.
3. Visualize the results of the model using plots or tables. For example, you can use the `matplotlib` library in Python or the `ggplot2` package in R to create a confusion

Step 2: Obtain a dataset of protein sequences and their known functions

Use a database such as the Protein Data Bank (PDB) or the UniProt database to obtain a dataset of protein sequences and their known functions. Preprocess the data by encoding the protein sequences as numerical data and splitting the data into training and testing sets.

Python

```
# Load the data into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('protein_data.csv')
# Encode the protein sequences as numerical data
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder() df['sequence'] =
le.fit_transform(df['sequence'])
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df['sequence'], df['function'], test_size=0.3)
```

R

```
# Load the data into a tibble library(tidyverse) df <-
read_csv('protein_data.csv') # Encode the protein sequences as
numerical data library(caret) df$sequence <- factor(df$sequence)
df$sequence <- as.integer(df$sequence) - 1 # Split the data into
training and testing sets set.seed(123) ind <-
createDataPartition(df$function, p=0.7, list=FALSE) train <-
df[ind,] test <- df[-ind,]
```

Step 3: Use machine learning algorithms to predict the functions of the proteins

Use the `scikit-learn` library in Python or the `caret` package in R to fit a model to the training data and make predictions on the testing data.

Python

```
# Fit a model to the training data
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier() model.fit(X_train, y_train)
# Make predictions on the testing data
y_pred = model.predict(X_test)
```

R

Copy code

```
# Fit a model to the training data
library(caret) model <- train(function ~ sequence, data=train,
method='rf') # Make predictions on the testing data y_pred <-
predict(model, test)
```

Step 4: Evaluate the performance of the model

Use evaluation metrics such as accuracy, precision, and recall to assess the performance of the model.

Python

```
# Calculate the accuracy of the model f
rom sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

```
# Calculate the precision of the model
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred, average='micro')
# Calculate the recall of the model
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred, average='micro')
```

R

Copy code

```
# Calculate the accuracy of the model library(caret) accuracy <-
confusionMatrix(y_pred, test$function)$overall[1] # Calculate the
precision of the model precision <- confusionMatrix(y_pred,
test$function)$byClass[1] # Calculate the recall of the model
recall <- confusionMatrix(y_pred, test$function)$byClass[5]
```

Step 5: Visualize the results of the model

Use plots or tables to visualize the results of the model.

Python

```
# Create a confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
# Visualize the confusion matrix using Matplotlib
import matplotlib.pyplot as plt
plt.imshow(cm, interpolation='nearest', cmap='Blues')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
# Add labels and title to the plot
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion matrix')
# Show the plot
plt.show()
```

R

```
# Create a confusion matrix
library(caret)
cm <- confusionMatrix(y_pred, test$function)

# Visualize the confusion matrix using ggplot2
library(ggplot2)
ggplot(cm$table, aes(x=Reference, y=Prediction)) +
```

```
  geom_tile(aes(fill=Freq)) +
  geom_text(aes(label=Freq), color='white') +
  labs(title='Confusion matrix’)
```

Submission

Please submit the following:

- Code script (Python or R)
- Output figures (e.g., confusion matrix)
- Written report describing your analysis and interpretation of the results
-