Hausarbeit_Nguyen

January 31, 2024

Name: Dang Quynh Tram Nguyen

Matrikelnummer: 5311561

Semester: Wintersemester 23/24

Modul: Data Science

1 Lesen Fasta Dateien

Die Daten der DNA-Fragmente der K562-Zelle, die als Enhancern und Promotoren bezeichnet sind, wurden von UCSC-Webseite heruntergeladen.

Promotoren sind Bestandteile von Genen und spielen eine entscheidende Rolle bei der Genregulation. Transkriptionsfaktoren sowie RNA-Polymerase binden an sie, um die Transkription der Gene zu initiieren. Enhancer sind spezifische DNA-Abschnitte, an die Transkriptionsfaktoren binden können. Sie optimieren die Transkriptionsaktivität der Gene, deren Promotoren mit ihnen gekoppelt sind.

[1]:
```
from Bio import SeqIO

# Funktion: Lesen Text-Dateien (Format ähnlich wie eine FASTA-Datei) und geben␣
 ↪eine Liste der Sequenzen zurück
def read_fasta(*paths):
    sequences = {'seq': []}
    for path in paths:
        with open (path, 'r') as fasta_file:
            for record in SeqIO.parse(fasta_file, 'fasta'):
                sequences['seq'].append(record.seq)
    return sequences
```

Die Anzahl der Enhancern nach dem Filtern ist ganz wenig. Deswegen wurden 3 Dateien mit insgesamt ca. 2000 Sequenzen gelesen. Die Datei von Promotoren enthält ca. 600 Sequenzen.

[2]:
```
enhancers = read_fasta('Enhancer.txt', 'Enhancer2.txt', 'Enhancer3.txt')
promotors = read_fasta('Promotor.txt')
```

[3]:
```
import pandas as pd

# Erstellen die Data Frames für die Sequenzendaten von Enhancers und Promotoren
```

```
df_enhancers = pd.DataFrame(enhancers)
df_promotors = pd.DataFrame(promotors)
```

[4]:
```
df_enhancers
```

[4]:
```
                                                    seq
0     (C, C, G, C, C, G, T, T, G, C, A, A, A, G, G, …
1     (G, A, T, T, C, A, T, G, G, C, T, G, A, A, A, …
2     (G, A, T, T, C, A, T, G, G, C, T, G, A, A, A, …
3     (G, A, T, T, C, A, T, G, G, C, T, G, A, A, A, …
4     (G, A, T, C, C, T, T, G, A, A, G, C, G, C, C, …
…                                                   …
2108  (G, A, T, C, C, A, C, C, C, A, C, C, T, T, G, …
2109  (G, A, T, C, C, A, C, C, C, A, C, C, T, T, G, …
2110  (G, A, T, C, C, A, C, C, C, A, C, C, T, T, G, …
2111  (G, A, T, C, C, A, C, C, C, A, C, C, T, T, G, …
2112  (G, A, T, C, C, A, C, C, C, A, C, C, T, T, G, …

[2113 rows x 1 columns]
```

[5]:
```
# Die Größen der Enhancern- und Promotoren-Daten
print(len(df_enhancers))
print(len(df_promotors))
```

```
2113
611
```

[6]:
```
# Sortieren die Duplikationen in den Daten aus
df_enhancers = df_enhancers.drop_duplicates(ignore_index=True)
df_promotors = df_promotors.drop_duplicates(ignore_index=True)

# Die Größen der Daten nach dem Aussortieren der Duplikationen
print(len(df_enhancers))
print(len(df_promotors))
```

```
227
385
```

[7]:
```
# Finden die überlappten Sequenzen von Enhancers und Promotoren
overlap_seq = df_enhancers[df_enhancers['seq'].isin(df_promotors['seq'])]
len(overlap_seq)
```

[7]:
```
111
```

[8]:
```
# Funktion: Sortieren die überlappten Sequenzen in beiden Daten aus
def df_without_overlap(df: pd.DataFrame):
    global overlap_seq
```

```
    # Mergen die Datenframe mit überlappten Sequenzen
    merged_df = pd.merge(df, overlap_seq, on='seq', how='outer', indicator=True)

    # Sortieren die Überlappten aus
    filtered_df = merged_df[merged_df['_merge'] == 'left_only'].
 ↪drop(columns='_merge')

    return filtered_df
```

[9]:
```
df_enhancers = df_without_overlap(df_enhancers)
df_promotors = df_without_overlap(df_promotors)
```

[10]:
```
# Die Größen der Daten nach dem Aussortieren der überlappten Sequenzen
print(len(df_enhancers))
print(len(df_promotors))
```

```
116
274
```

2 Hypothesentest: Unterschied zwischen GC-Anteil in Enhancern und Promotoren

Der GC-Anteil ist ein charakteristisches Merkmal von Nukleinsäuremolekülen und gibt den Gehalt an Guanin (G) und Cytosin (C) im Verhältnis zu den gesamten Nukleobasen in einer Sequenz an. In diesem Abschnitt wurden die GC-Anteile der Enhancer und Promotoren berechnet und miteinander verglichen, um festzustellen, ob signifikante Unterschiede vorhanden sind.

2.1 Vorbereiten der GC-Anteilen

[11]:
```
# Funktion: Berechnung die GC-Anteil in jeder Sequenzen
# Formel: Summe der Anzahl von C und G über Summe aller Nukleotiden in der␣
 ↪Sequenz
def add_gc_amount_col(df: pd.DataFrame):
    df['gc_amount'] = df.apply(lambda row: (row['seq'].count('C') + row['seq'].
 ↪count('G')) / len(row['seq']), axis = 1)
```

[12]:
```
add_gc_amount_col(df_enhancers)
add_gc_amount_col(df_promotors)
```

[13]:
```
df_enhancers
```

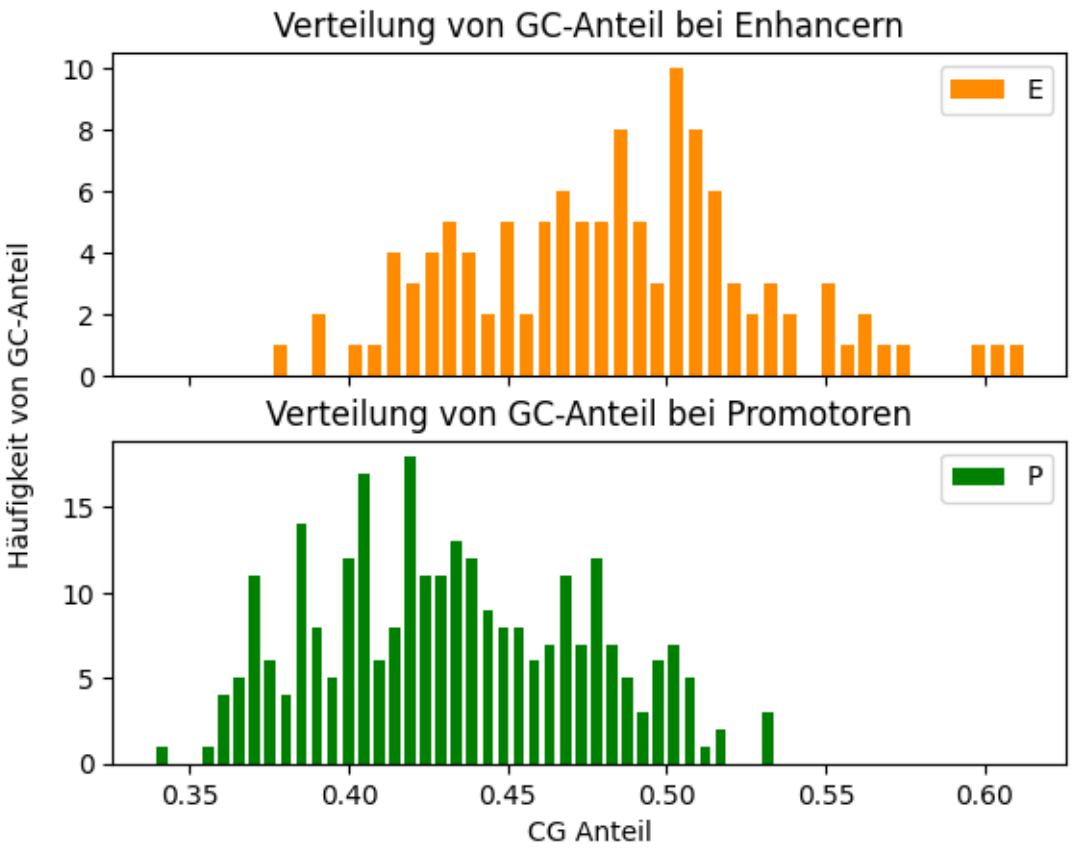
[13]:

	seq	gc_amount
0	(C, C, G, C, C, G, T, T, G, C, A, A, A, G, G, …	0.429431
1	(G, A, T, T, C, A, T, G, G, C, T, G, A, A, A, …	0.391445
2	(G, A, T, C, C, T, T, G, A, A, G, C, G, C, C, …	0.428906
14	(C, T, G, C, C, C, T, T, G, C, T, G, A, C, C, …	0.564309

```
15   (C, A, G, T, C, C, C, A, G, C, G, G, A, C, A, …   0.572551
..                                                  …        …
202  (G, C, T, T, C, C, C, A, A, A, C, T, G, C, T, …   0.504595
209  (C, A, T, C, A, C, C, A, C, A, A, T, C, A, A, …   0.444295
213  (G, A, T, C, C, C, C, A, T, A, T, T, C, C, C, …   0.511536
217  (G, A, A, T, T, C, T, T, T, G, T, A, A, T, A, …   0.415281
225  (A, T, C, T, T, T, T, C, A, G, C, T, T, A, G, …   0.476236

[116 rows x 2 columns]
```

2.2 Visualisieren die Verteilung der GC-Anteile bei Enhancers und Promotoren

[14]:
```
import matplotlib.pyplot as plt

# Erstellen die Plots, da sie die x-Achse (GC-Anteil) miteinander teilen
fig, axs = plt.subplots(2,1, sharex = True)

axs[0].hist(df_enhancers['gc_amount'], bins = 40, rwidth = 0.7, color =␣
 ↪'darkorange')
axs[0].legend('Enhancern')
axs[1].hist(df_promotors['gc_amount'], bins = 40, rwidth = 0.7, color = 'green')
axs[1].legend('Promotoren')

# Setzen die Labels der Achsen
fig.text(0.04, 0.5,'Häufigkeit von GC-Anteil', va='center', rotation =␣
 ↪'vertical')
plt.xlabel('CG Anteil')

# Beschriften der Plots
axs[0].set_title('Verteilung von GC-Anteil bei Enhancern')
axs[1].set_title('Verteilung von GC-Anteil bei Promotoren')

plt.show()
```



2.3 Testen die Normalveteilung von jeder Stichprobe durch Shapiro Wilk Test

Stichproben: Die GC-Anteile der Enhancern und Promotoren

Nullhypothese: Normalverteilung

Signifikant Level = 0.05

[15]:
```
from scipy import stats

sig_lv = 0.05

e_stat_sw, e_p_val_sw = stats.shapiro(df_enhancers['gc_amount'])
p_stat_sw, p_p_val_sw = stats.shapiro(df_promotors['gc_amount'])

print('Shapiro-Wilk Test:')
if e_p_val_sw >= sig_lv:
    print(f'Die GC-Anteil bei Enhancern verteilt normal (p-value =␣
 ↪{e_p_val_sw})')
else:
```

```
    print(f'Die GC-Anteil bei Enhancern verteilt NICHT normal (p-value =␣
 ↪{e_p_val_sw})')

if p_p_val_sw >= sig_lv:
    print(f'Die GC-Anteil bei Promotoren verteilt normal (p-value =␣
 ↪{p_p_val_sw})')
else:
    print(f'Die GC-Anteil bei Promotoren verteilt NICHT normal (p-value =␣
 ↪{p_p_val_sw})')
```

```
Shapiro-Wilk Test:
Die GC-Anteil bei Enhancern verteilt normal (p-value = 0.4938776195049286)
Die GC-Anteil bei Promotoren verteilt NICHT normal (p-value =
0.0018061109585687518)
```

2.4 Testen die Varianzhomogenität von jeder Sitchprobe durch Levene Test

Stichproben: Die GC-Anteile der Enhancern und Promotoren

Nullhypothese: Varianzhomogenität

Signifikant Level = 0.05

[16]:
```
sig_lv = .05

stat_le, p_val_le = stats.levene(df_enhancers['gc_amount'],␣
 ↪df_promotors['gc_amount'])

if p_val_le >= sig_lv:
    print(f'2 Stichproben sind varianzhomogen (p-value = {p_val_le})')
else:
    print(f'2 Stichproben sind NICHT varianzhomogen (p-value = {p_val_le})')
```

```
2 Stichproben sind varianzhomogen (p-value = 0.35227239234991037)
```

2.5 Testen die gleiche GC-Anteile zwischen beiden Stichproben

2 unabhängigen Stichproben (GC-Anteile bei Enhancern und Promotoren) sind varianzhomogen, aber nur eine ist normal verteilt

=> Wählen Mann-Whitney-U Test (weder Normalverteilung noch Varianzhomogenität benötigt)

Nullhypothese: gleiche GC-Anteile bei Enhancern und Promotoren

Signifikant Level = 0.05

[17]:
```
sig_lv = 0.05

stat_man, p_val_man = stats.mannwhitneyu(df_enhancers['gc_amount'],␣
 ↪df_promotors['gc_amount'], alternative = 'two-sided')
```

```
if p_val_man >= sig_lv:
    print(f'Enhancern und Promotoren haben gleiche Verteilung der GC-Anteil␣
 ↪(p-value = {p_val_man})')
else:
    print(f'Die Verteilungen der GC-Anteil der Enhancern und Promotoren sind␣
 ↪signifikant (p-value = {p_val_man})')
```

```
Die Verteilungen der GC-Anteil der Enhancern und Promotoren sind signifikant
(p-value = 1.8437876883184977e-19)
```

3 Klassifikation: Unterschied zwischen Enhancern Promotoren anhand 2 meren Nukleotiden

In diesem Abschnitt werden die Anteile der Dinukleotide gezählt. Diese dienen als Merkmale, die den Sequenztyp repräsentieren, unabhängig davon, ob es sich um einen Enhancer oder einen Promotor handelt. Die Klassifikation erfolgt mithilfe von Random Forest- und KNN-Modellen. Diese Modelle werden verwendet, um vorherzusagen, ob eine Sequenz als Enhancer oder Promotor klassifiziert werden kann.

3.1 Vorbereiten der Dinukleotiden-Anteile (Anteile der 2-mere-Nukleotiden)

[18]:
```
# Erstellen die Liste von allen möglichen 2-meren Nukleotiden
_2mer = []
nu = ['A', 'T', 'G', 'C']
for i in nu:
    _2mer.extend([i + n for n in nu])

_2mer
```

[18]:
```
['AA',
 'AT',
 'AG',
 'AC',
 'TA',
 'TT',
 'TG',
 'TC',
 'GA',
 'GT',
 'GG',
 'GC',
 'CA',
 'CT',
 'CG',
 'CC']
```

[19]:
```
# Funktion: Berechnen die Anteile alle 2-meren Nukleotiden in jeder Sequenzen
# Formel für Anzahl möglichen Nukleotidenpaare in einer Sequenz = n - 2 + 1 (n␣
 ↪ist die Anzahl der Nukleotide in dieser Sequenz)
def add_col_for_2mer(df: pd.DataFrame):
    for pair in _2mer:
        df[pair] = df.apply(lambda row: row['seq'].count(pair) /␣
 ↪(len(row['seq']) - 2 + 1), axis = 1)
```

[20]:
```
add_col_for_2mer(df_enhancers)
add_col_for_2mer(df_promotors)
```

[21]:
```
# Addieren Klasse-Spalte, um die Daten zu unterscheiden, wenn beide Dataframes␣
 ↪verkettet werden
df_enhancers['class'] = 'enhancer'
df_promotors['class'] = 'promotor'
```

[22]:
```
# Verketten beide Dataframes von Enhancern und Promotoren
combine_df_2mer = pd.concat([df_enhancers, df_promotors], ignore_index= True)
combine_df_2mer
```

[22]:

	seq	gc_amount	AA	\
0	(C, C, G, C, C, G, T, T, G, C, A, A, A, G, G, …	0.429431	0.072356	
1	(G, A, T, T, C, A, T, G, G, C, T, G, A, A, A, …	0.391445	0.078460	
2	(G, A, T, C, C, T, T, G, A, A, G, C, G, C, C, …	0.428906	0.073188	
3	(C, T, G, C, C, C, T, T, G, C, T, G, A, C, C, …	0.564309	0.038431	
4	(C, A, G, T, C, C, C, A, G, C, G, G, A, C, A, …	0.572551	0.027586	
..	…	…	…	
385	(G, A, T, G, G, G, C, C, C, C, T, G, T, A, G, …	0.497992	0.052668	
386	(A, G, C, C, C, T, C, C, C, C, C, C, A, C, T, …	0.508756	0.045737	
387	(C, T, C, C, A, T, G, T, G, G, T, G, A, C, A, …	0.476493	0.057797	
388	(T, C, A, C, C, C, C, A, G, C, T, T, G, T, T, …	0.501508	0.052304	
389	(C, A, G, A, G, C, C, T, T, A, A, G, C, A, A, …	0.513124	0.046016	

	AT	AG	AC	TA	TT	TG	TC	\
0	0.072685	0.073117	0.053051	0.061073	0.059268	0.069934	0.058998	
1	0.082852	0.069452	0.053150	0.070495	0.066748	0.072306	0.059577	
2	0.074309	0.072103	0.058585	0.060261	0.055277	0.068589	0.058925	
3	0.036582	0.075059	0.049518	0.025595	0.040976	0.081462	0.058854	
4	0.031163	0.071236	0.043123	0.026137	0.050717	0.088858	0.062831	
..	…	…	…	…	…	…	…	
385	0.050668	0.076805	0.050737	0.043133	0.052113	0.071320	0.064838	
386	0.047901	0.079671	0.047476	0.040588	0.052834	0.077365	0.063712	
387	0.056616	0.075753	0.051882	0.049217	0.055104	0.072993	0.060283	
388	0.051067	0.080967	0.049917	0.042606	0.049824	0.072506	0.063768	
389	0.045947	0.080431	0.046590	0.038715	0.053097	0.074907	0.065405	

	GA	GT	GG	GC	CA	CT	CG	\

```
0    0.060701  0.046259  0.045030  0.047769  0.077080  0.071061  0.011679
1    0.060743  0.049205  0.036252  0.039205  0.074217  0.070346  0.007370
2    0.062242  0.046859  0.044133  0.044573  0.082494  0.066608  0.012981
3    0.060306  0.054942  0.074716  0.076144  0.075258  0.074387  0.034871
4    0.052604  0.072179  0.081484  0.071060  0.066781  0.074484  0.035772
..        …         …         …         …         …         …
385  0.059874  0.047434  0.057996  0.058325  0.075203  0.081190  0.017502
386  0.061252  0.050775  0.061609  0.062090  0.073202  0.082989  0.017081
387  0.060179  0.048527  0.054509  0.056682  0.074855  0.077350  0.016642
388  0.062783  0.048354  0.059719  0.059054  0.076567  0.079453  0.016719
389  0.063659  0.049692  0.063583  0.060783  0.070594  0.083388  0.018796

           CC     class
0    0.045593  enhancer
1    0.037518  enhancer
2    0.044869  enhancer
3    0.066831  enhancer
4    0.068822  enhancer
..        …       …
385  0.063167  promotor
386  0.062115  promotor
387  0.056711  promotor
388  0.061559  promotor
389  0.063022  promotor

[390 rows x 19 columns]
```

[23]:
```
# Halten nur die Spalten über 2-mere Nukleotid und Klasse
combine_df_2mer = combine_df_2mer[_2mer + ['class']]
combine_df_2mer
```

[23]:
```
           AA        AT        AG        AC        TA        TT        TG  \
0    0.072356  0.072685  0.073117  0.053051  0.061073  0.059268  0.069934
1    0.078460  0.082852  0.069452  0.053150  0.070495  0.066748  0.072306
2    0.073188  0.074309  0.072103  0.058585  0.060261  0.055277  0.068589
3    0.038431  0.036582  0.075059  0.049518  0.025595  0.040976  0.081462
4    0.027586  0.031163  0.071236  0.043123  0.026137  0.050717  0.088858
..        …         …         …         …         …         …
385  0.052668  0.050668  0.076805  0.050737  0.043133  0.052113  0.071320
386  0.045737  0.047901  0.079671  0.047476  0.040588  0.052834  0.077365
387  0.057797  0.056616  0.075753  0.051882  0.049217  0.055104  0.072993
388  0.052304  0.051067  0.080967  0.049917  0.042606  0.049824  0.072506
389  0.046016  0.045947  0.080431  0.046590  0.038715  0.053097  0.074907

           TC        GA        GT        GG        GC        CA        CT  \
0    0.058998  0.060701  0.046259  0.045030  0.047769  0.077080  0.071061
1    0.059577  0.060743  0.049205  0.036252  0.039205  0.074217  0.070346
```

```
2    0.058925  0.062242  0.046859  0.044133  0.044573  0.082494  0.066608
3    0.058854  0.060306  0.054942  0.074716  0.076144  0.075258  0.074387
4    0.062831  0.052604  0.072179  0.081484  0.071060  0.066781  0.074484
..        …         …         …         …         …         …         …
385  0.064838  0.059874  0.047434  0.057996  0.058325  0.075203  0.081190
386  0.063712  0.061252  0.050775  0.061609  0.062090  0.073202  0.082989
387  0.060283  0.060179  0.048527  0.054509  0.056682  0.074855  0.077350
388  0.063768  0.062783  0.048354  0.059719  0.059054  0.076567  0.079453
389  0.065405  0.063659  0.049692  0.063583  0.060783  0.070594  0.083388

           CG        CC     class
0    0.011679  0.045593  enhancer
1    0.007370  0.037518  enhancer
2    0.012981  0.044869  enhancer
3    0.034871  0.066831  enhancer
4    0.035772  0.068822  enhancer
..        …         …         …
385  0.017502  0.063167  promotor
386  0.017081  0.062115  promotor
387  0.016642  0.056711  promotor
388  0.016719  0.061559  promotor
389  0.018796  0.063022  promotor

[390 rows x 17 columns]
```

3.2 Klassifikation mit Random Forest

Die Standardparameter für Entscheidungsbäume im Random Forest wurden standardmäßig beibehalten. Die Anzahl der Bäume für den Wald (n_estimators) beträgt 50.

[24]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Daten für das Training aufteilen
# Anteil der Daten für Training und Test Data 80:20
# Random_state: speichert den zufälligen Zustand, damit das Ergebnis nach␣
 ↪mehrmale Ausführung nicht geändert wird
# Shuffle Daten vor der Aufteilung
X_train, X_test, y_train, y_test = train_test_split(combine_df_2mer.loc[:,␣
 ↪combine_df_2mer.columns != 'class'], combine_df_2mer['class'], test_size = 0.
 ↪2, random_state = 13, shuffle = True)

# Erstellen Klassifikator
classifier = RandomForestClassifier(n_estimators=50, random_state = 13)

# Trainieren Klassifikator
classifier.fit(X_train, y_train)
```

[24]:
```
RandomForestClassifier(n_estimators=50, random_state=13)
```

[25]:
```
# Vorhersagen auf Testdaten
y_pred_rd = classifier.predict(X_test)
```

[26]:
```
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪ConfusionMatrixDisplay, classification_report
```

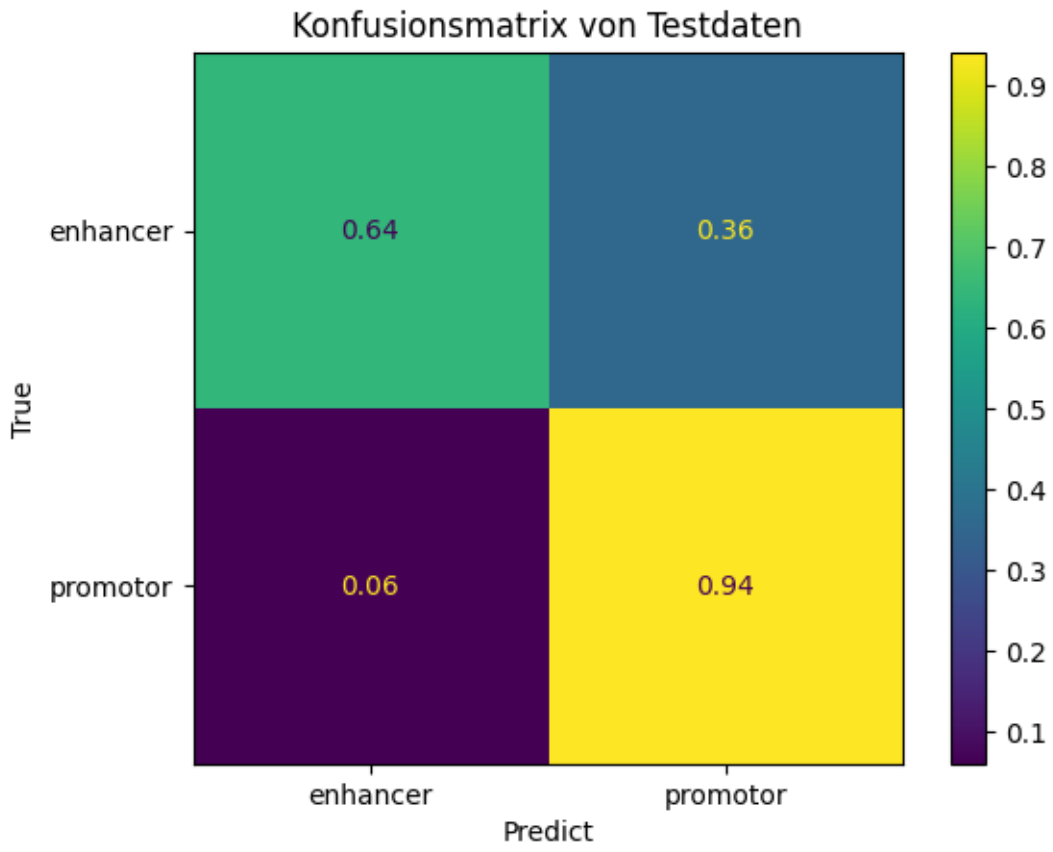
[27]:
```
# Auswerten die Genauigkeit
accuracy_rd = accuracy_score(y_test, y_pred_rd)
print(f'Accuracy: {accuracy_rd}')
```

```
Accuracy: 0.8333333333333334
```

[28]:
```
# Konfusionsmatrix
conf_matrix_rd = confusion_matrix(y_test, y_pred_rd, normalize='true')
```

[29]:
```
# Visualisieren die Konfusionsmatrizen
def display_conf_matrix(confusion_matrix):
    ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,␣
↪display_labels=classifier.classes_).plot()
    plt.title(f'Konfusionsmatrix von Testdaten')
    plt.xlabel('Predict')
    plt.ylabel('True')
    plt.show()

print(conf_matrix_rd)
display_conf_matrix(conf_matrix_rd)
```

```
[[0.64285714 0.35714286]
 [0.06       0.94      ]]
```



[30]:
```
report_rd = classification_report(y_test, y_pred_rd)
print(report_rd)
```

```
              precision    recall  f1-score   support

    enhancer       0.86      0.64      0.73        28
    promotor       0.82      0.94      0.88        50

    accuracy                           0.83        78
   macro avg       0.84      0.79      0.81        78
weighted avg       0.84      0.83      0.83        78
```

Fazit von Random Forest:

Die Genauigkeit ist ganz hoch (83 %).

Der Recall von 0.94 sowie die Percision von 0.82 an der Promotoren-Klasse weist darauf hin, dass das Modell Promotoren gut erkennen kann. Enhancer wurden mit einer Genauigkeit von 86 % erkannt, was darauf hinweist, dass das Modell Enhancer gut vorhersagen kann. Allerdings wurden nur 64 % der insgesamt vorhandenen Enhancer identifiziert. Dies legt nahe, dass dem Modell möglicherweise immer noch Enhancer entgehen können.

3.3 Klassifikation mit KNN (K-Nearest Neigbors)

[31]:
```
from sklearn.neighbors import KNeighborsClassifier

# Erstellen KneighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=5) # parameter n_neighbors =␣
 ↪5 default

# Trainieren classifier
knn_classifier.fit(X_train, y_train)

# Make prediction
y_pred_knn = knn_classifier.predict(X_test)

accuracy_knn = accuracy_score(y_test, y_pred_knn)
print('Accuracy score =', accuracy_knn)
```

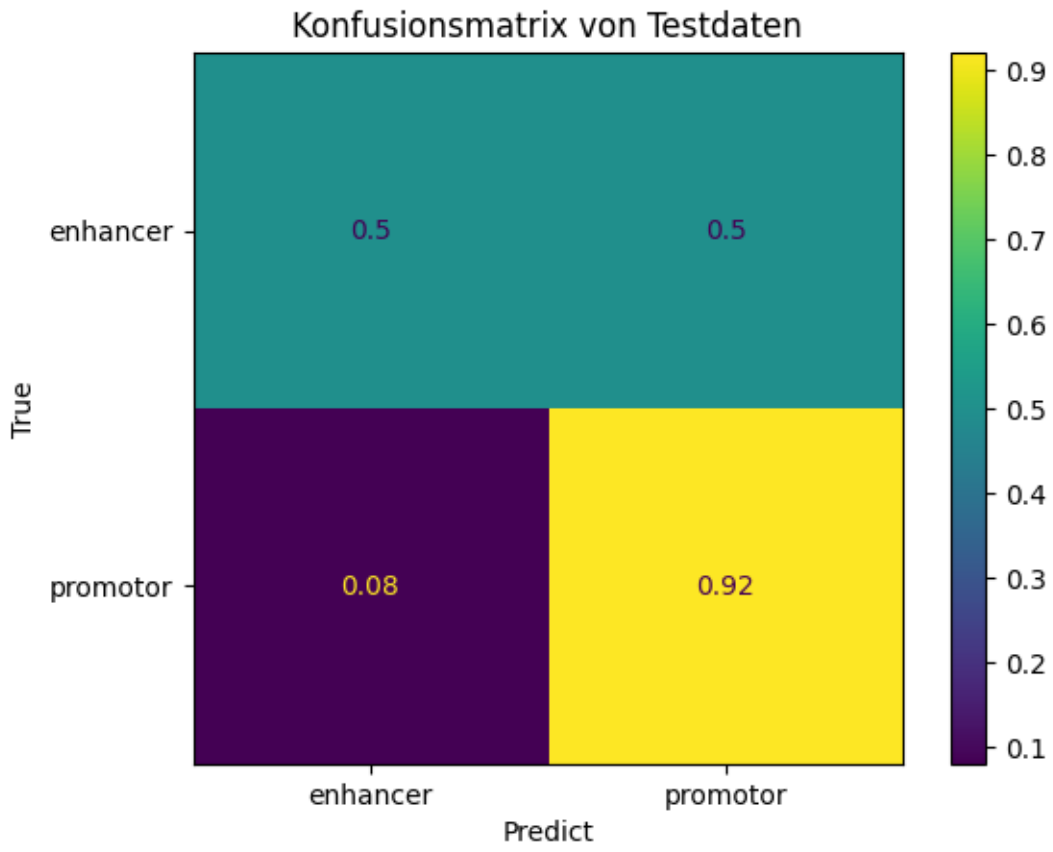
```
Accuracy score = 0.7692307692307693
```

```
C:\Users\quynh\anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

[32]:
```
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn, normalize='true')
conf_matrix_knn
```

[32]:
```
array([[0.5 , 0.5 ],
       [0.08, 0.92]])
```

[33]:
```
display_conf_matrix(conf_matrix_knn)
```



[34]:
```
report_knn = classification_report(y_test, y_pred_knn)
print(report_knn)
```

```
              precision    recall  f1-score   support

    enhancer       0.78      0.50      0.61        28
    promotor       0.77      0.92      0.84        50

    accuracy                           0.77        78
   macro avg       0.77      0.71      0.72        78
weighted avg       0.77      0.77      0.75        78
```

Fazit von KNN:

Die Genauigkeit ist gering (0.77) im Vergleich zum Random Forest Modell

Der höhere Recall (0,92) für die Promotoren-Klasse zeigt, dass das KNN-Modell Promotoren gut erkennen kann. Im Gegensatz dazu beträgt der Recall für die Enhancer-Klasse 0,5, was darauf hindeutet, dass das Modell bei der korrekten Erkennung von Enhancern nicht so gut abschneidet. Die Präzision für beide Klassen liegt ebenfalls unter 80 % (78 % und 77 %), was bedeutet, dass es relativ wenige falsch positive Vorhersagen gibt.

Zusammenfassung:

Die Verwendung der Anteile der Dinukleotide als Merkmale für zwei Klassen, nämlich Enhancer und Promotoren, zeigt, dass der Random Forest bessere Vorhersagen im Vergleich zur KNN-Klassifikation treffen kann. Dennoch ist es notwendig, dass beide Modelle ihre Präzision verbessern. Die Merkmale, die zur Unterscheidung der beiden Klassen verwendet werden (2-mere Nukleotide), sind möglicherweise nicht optimal und könnten das Training der Modelle beeinträchtigen.

4 Längenverteilung von Enhancers und Promotoren

4.1 Lesen CSV-Datei

- Die Datei beinhaltet die Infos der verbundenen/gepaaren Enhancern und Promotoren
- Halten nur 1000 ersten Zeile der Daten

[35]:
```
df_pair = pd.read_csv('pairs.csv')
df_pair = df_pair.iloc[:1000]
```

4.2 Berechnen die Längen der Enhancern und Promotoren

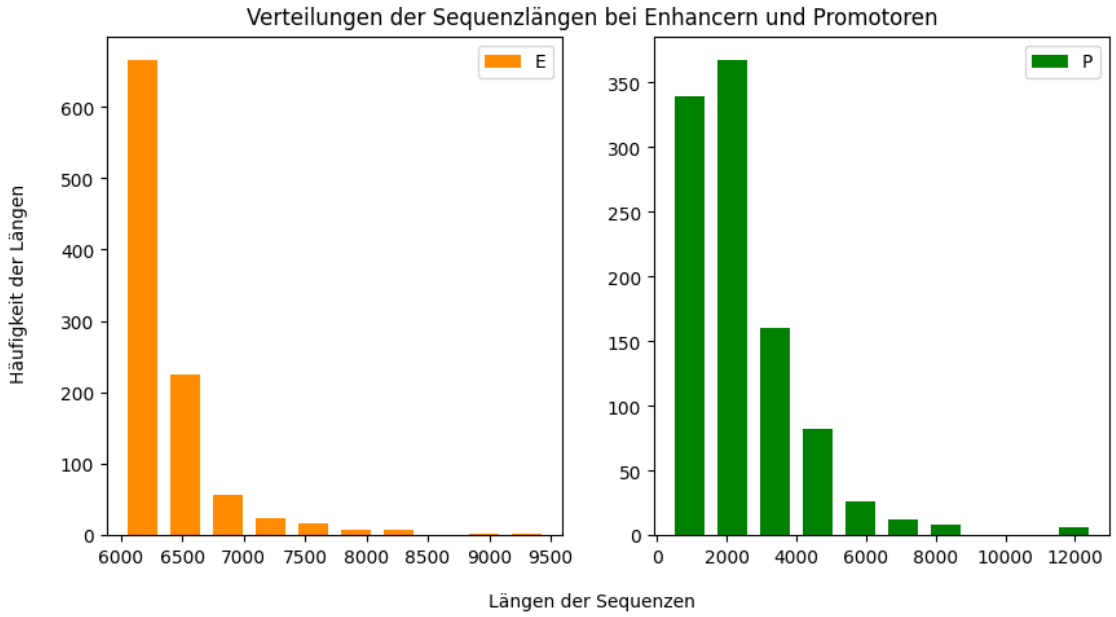
[36]:
```
# Erstellen die Listen, die die Längen der Enhancern und ihrer entsprechenden␣
 ↪Promotoren enthalten
len_en = []
len_pr = []
for i in range(len(df_pair)):
    len_en.append(df_pair.iloc[i]['enhancer_end'] - df_pair.
 ↪iloc[i]['enhancer_start'])
    len_pr.append(df_pair.iloc[i]['promoter_end'] - df_pair.
 ↪iloc[i]['promoter_start'])
```

4.3 Visualisieren die Daten

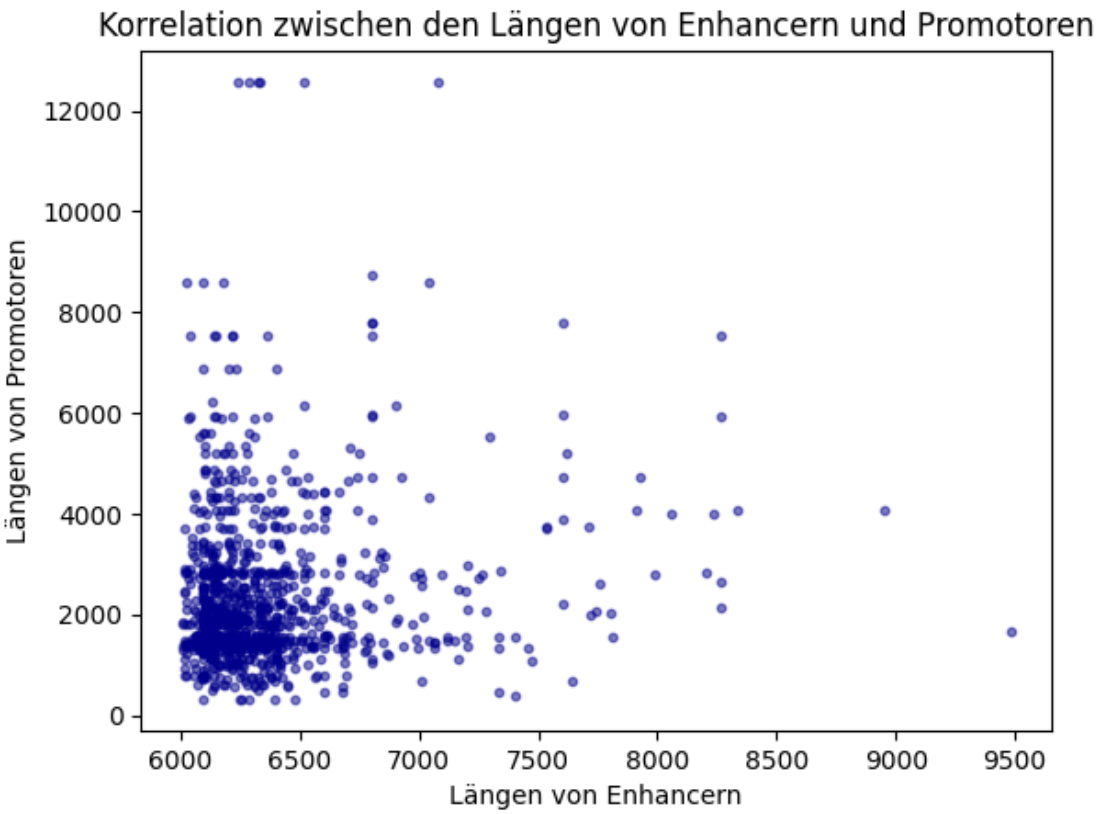
- Die Verteilungen die Sequenzlängen bei Enhancern und Promotoren
- Scatterplot von Sequenzlängen bei den gepaaren Enhancern und Promotoren

[37]:
```
# Erstellen die Plots der Verteilungen von Sequenzlängen bei Enhancern und␣
 ↪Promotoren
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].hist(len_en, rwidth = 0.7, color = 'darkorange')
axs[0].legend('Enhancern')
axs[1].hist(len_pr, rwidth = 0.7, color = 'green')
axs[1].legend('Promotoren')

# Setzen die Labels der Achsen und beschriften die Plots
fig.text(0.5, 0,'Längen der Sequenzen', ha='center')
fig.text(0.05, 0.5,'Häufigkeit der Längen', va='center', rotation = 'vertical')
```

```
fig.text(0.5, 0.9, 'Verteilungen der Sequenzlängen bei Enhancern und␣
 ↪Promotoren', ha='center', fontsize='large')

plt.show()
```



[38]:
```
plt.scatter(len_en, len_pr, color='darkblue', marker='o',s=10, alpha=0.5)
plt.xlabel('Längen von Enhancern')
plt.ylabel('Längen von Promotoren')
plt.title('Korrelation zwischen den Längen von Enhancern und Promotoren')
plt.show()
```



4.4 Korrelation zwischen den Längenverteilungen von Enhancern und Promotoren

Anhand des obigen Scatterplot sieht man keine lineare Zusammenhang zwischen die Längen von Enhancern und Promotoren. Die monotone Beziehung ist aber auch nicht deutlich.

=> Wählen zwei Korrelationsmethoden: Spearman und Kendall-Tau, um did monotone Beziehung zwischen beiden Stichproben zu sowie ihre Korrelation zu bewerten

Korrelationskoeffizient in [-1, 1]: - geht nah zu 0 => keine Korrelation - geht nah zur Grenze => negative/positive Korrelation

Nullhypothese: Keine Korrelation zwischen Sequenzlängen von Enhancern und Promotoren

Signifikantslevel = 0.05

[39]:
```
# Korrelationskoeffizienzen (Pearson, Spearman, Kendall-Tau)
spea_cor_coef, spea_p_val = stats.spearmanr(len_en, len_pr)
ken_cor_coef, ken_p_val = stats.kendalltau(len_en, len_pr)
```

[40]:
```
# Interpretieren die Ergebnisse
sig_lv = 0.05
```

```
cor_method = {'Spearman': [spea_cor_coef, spea_p_val],
              'Kendall-Tau': [ken_cor_coef, ken_p_val]}

for method in cor_method.keys():
    print(method)
    print(f'  Korrelationskoeffizient = {cor_method[method][0]}')
    if cor_method[method][0] <= -0.5:
        print('  Negative Korrelation')
    elif cor_method[method][0] >= 0.5:
        print('  Positive Korrelation')
    else:
        print('  Keine Korrelation')


    if cor_method[method][1] >= sig_lv:
        print(f'  Nullhypothese ist angenommen ({cor_method[method][1]})')
    else: print(f'  Nullhypothese ist abgelehnt ({cor_method[method][1]})')
    print()
```

```
Spearman
  Korrelationskoeffizient = 0.052520387479355786
  Keine Korrelation
  Nullhypothese ist angenommen (0.09693277628700754)

Kendall-Tau
  Korrelationskoeffizient = 0.03466580795127028
  Keine Korrelation
  Nullhypothese ist angenommen (0.10226778673150758)
```

Aus den beiden Koeffizienten (0.05 und 0.03) lässt sich schließen, dass die Verteilungen der Sequenzlängen von Enhancern und Promotoren nicht miteinander korrelieren.

5 Anzahl der Enhancern/ Promotoren, die an jedem Promotor/Enhancer paaren

5.1 Lesen CSV-Datei, die Enhancern und Promotoren in Paar enthält

[41]:
```
df_pair = pd.read_csv('pairs.csv')
```

[42]:
```
#Wählen nur Paaren in Chromosomen 1
new_df_pair = df_pair[df_pair['enhancer_chrom'].isin(['chr1'])]
```

5.2 Zahlen die gepaarten Promotoren/Enhancern an jedem Enhancer/Promotor

[43]:
```
promoters_per_enhancer = new_df_pair[['enhancer_name', 'promoter_name']].
↪groupby(by='enhancer_name').size().reset_index(name='amount_pairs')
promoters_per_enhancer
```

[43]:

	enhancer_name	amount_pairs
0	K562|chr1:100059884-100059989	1
1	K562|chr1:100066109-100066571	1
2	K562|chr1:100111245-100111600	1
3	K562|chr1:100113600-100113726	2
4	K562|chr1:100236658-100237667	1
…	…	…
5126	K562|chr1:9953367-9953551	2
5127	K562|chr1:997383-997594	3
5128	K562|chr1:997686-997800	3
5129	K562|chr1:9988982-9990484	1
5130	K562|chr1:99968907-99969204	1

[5131 rows x 2 columns]

[44]:
```
enhancers_per_promoter = new_df_pair[['enhancer_name', 'promoter_name']].
↪groupby(by='promoter_name').size().reset_index(name='amount_pairs')
enhancers_per_promoter
```

[44]:

	promoter_name	amount_pairs
0	K562|chr1:10002482-10004387	9
1	K562|chr1:100230231-100232351	2
2	K562|chr1:100314998-100316621	5
3	K562|chr1:100434853-100436345	7
4	K562|chr1:100502303-100505046	4
..	…	…
827	K562|chr1:9711708-9712591	15
828	K562|chr1:9747015-9748363	7
829	K562|chr1:9884215-9884662	6
830	K562|chr1:994308-995511	13
831	K562|chr1:9969248-9970422	6

[832 rows x 2 columns]

5.3 Berechnen die statistischen Maße

[45]:
```
# Funktion: Berechnen minimale, maximale und durchschnittliche Anzahl der␣
↪Enhancern/Promotoren, die mit einem Promoter/Enhancer gepaart sind
def statistical_measures(amount_pair: pd.DataFrame, seq_type: str):
    max_ = amount_pair['amount_pairs'].max()
```

```
    min_ = amount_pair['amount_pairs'].min()
    mean_ = amount_pair['amount_pairs'].mean()
    pair_type = 'Promotoren' if seq_type == 'Enhancer' else 'Enhancern'
    print(f'Maximale Anzahl der gepaarten {pair_type} bei einem {seq_type}:␣
↪{max_}')
    print(f'Minimale Anzahl der gepaarten {pair_type} bei einem {seq_type}:␣
↪{min_}')
    print(f'Durchschnitliche Anzahl der gepaarten {pair_type} bei einem␣
↪{seq_type}: {round(mean_, 2)}')
```

[46]:
```
statistical_measures(promoters_per_enhancer, 'Enhancer')
```

```
Maximale Anzahl der gepaarten Promotoren bei einem Enhancer: 8
Minimale Anzahl der gepaarten Promotoren bei einem Enhancer: 1
Durchschnitliche Anzahl der gepaarten Promotoren bei einem Enhancer: 1.53
```

[47]:
```
statistical_measures(enhancers_per_promoter, 'Promotor')
```

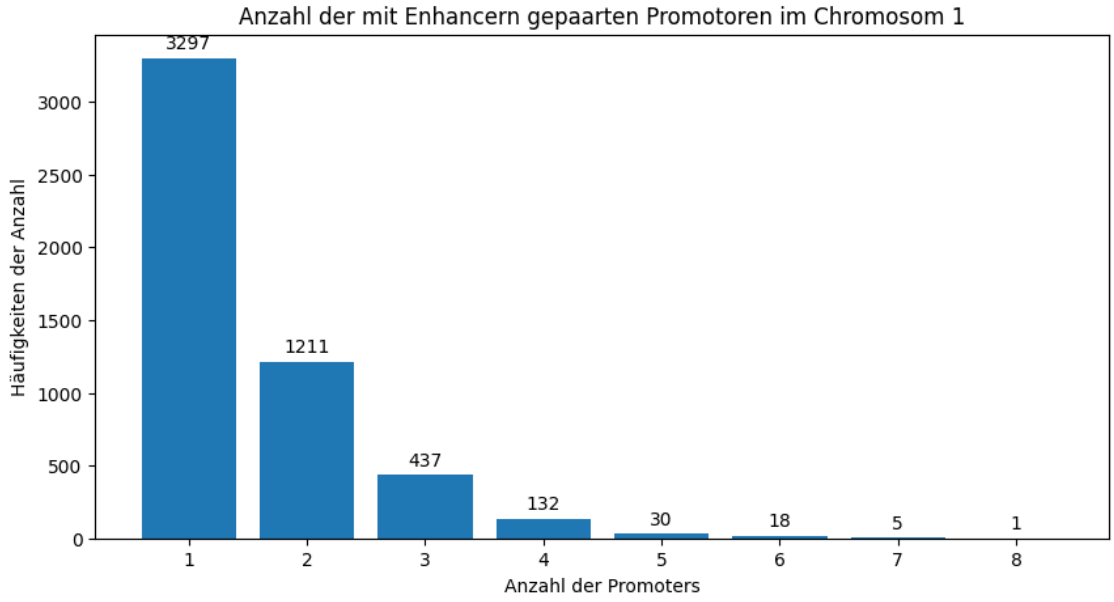
```
Maximale Anzahl der gepaarten Enhancern bei einem Promotor: 35
Minimale Anzahl der gepaarten Enhancern bei einem Promotor: 1
Durchschnitliche Anzahl der gepaarten Enhancern bei einem Promotor: 9.45
```

5.4 Visualisieren die Häufigkeiten von Anzahl der gepaarten Enhancern/Promotoren

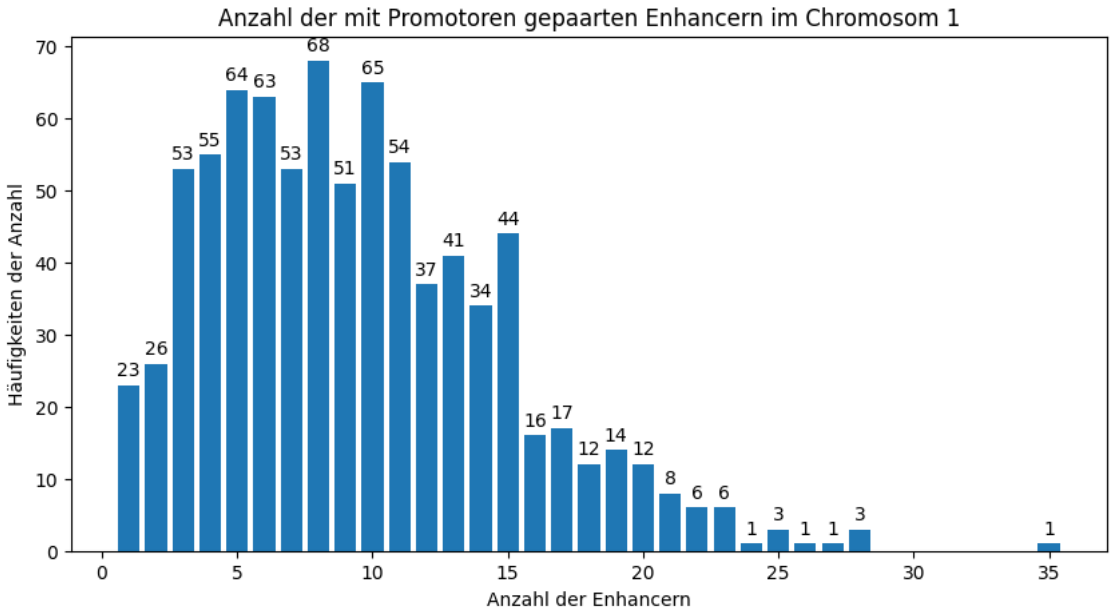
[48]:
```
# Funktion: Erstellen den Plot über die Verteilung der Anzahl der gepaarten␣
↪Enhancern/Promotoren
def frequency_amount_plot(amount_pairs: pd.DataFrame, seq_type: str):
    # Zahlen die Häufigkeiten der Anzahl
    frequency_amount = amount_pairs.groupby(by='amount_pairs').size().
↪reset_index(name='count')
    #print(frequency_amount)

    # Erstellen den Plot
    plt.figure(figsize=(10,5))

    bars = plt.bar(frequency_amount['amount_pairs'], frequency_amount['count'])

    # Addieren die Werten auf den Bars
    plt.bar_label(bars, padding=3)

    # Setzen die Achsen-Beschriftungen und den Titel
    if seq_type == 'Enhancer':
        plt.xlabel('Anzahl der Promoters')
        plt.ylabel('Häufigkeiten der Anzahl')
        plt.title('Anzahl der mit Enhancern gepaarten Promotoren im Chromosom␣
↪1')
```

```
    else:
        plt.xlabel('Anzahl der Enhancern')
        plt.ylabel('Häufigkeiten der Anzahl')
        plt.title('Anzahl der mit Promotoren gepaarten Enhancern im Chromosom␣
↪1')
    plt.show()
```

[49]:
```
frequency_amount_plot(promoters_per_enhancer, 'Enhancer')
```



[50]:
```
frequency_amount_plot(enhancers_per_promoter, 'Promotor')
```



Statistiken und Diagramme zeigen, dass oft ein Promoter durch einen Enhancer verstärkt wird. Umgekehrt kann ein Enhancer auch mehrere Promotoren unterstützen.

6 Verteilungen der Enhancern und Promotoren auf den unterschiedlichen Chromosomen

6.1 Filtern und zahlen die Enhancern/Promotoren in jedem Chromosom

[51]:
```
# Filtern zahlen die Enhancern in jedem Chromosom sowie entfernen die␣
 ↪Duplikationen
df_en_per_chr = df_pair[['enhancer_chrom', 'enhancer_name']].
 ↪drop_duplicates(ignore_index=True)
print(df_en_per_chr)
enhancers_count_per_chr = df_en_per_chr['enhancer_chrom'].value_counts()
enhancers_count_per_chr
```

```
      enhancer_chrom                 enhancer_name
0               chr1     K562|chr1:6454864-6455189
1               chr1     K562|chr1:6457976-6458177
2               chr1     K562|chr1:9935400-9935544
3               chr1   K562|chr1:16399567-16400081
4               chr1   K562|chr1:16400449-16400658
…                …                            …
37046          chr19  K562|chr19:44010200-44010280
37047           chr7     K562|chr7:5863365-5863600
37048          chr10  K562|chr10:94330214-94330314
```

```
37049          chr12  K562|chr12:95640200-95640262
37050          chr15  K562|chr15:74268181-74268335

[37051 rows x 2 columns]
```

[51]:
```
chr1     5131
chr19    2868
chr6     2707
chr17    2422
chr11    2229
chr2     2207
chr7     2008
chr3     2003
chr12    1969
chr9     1508
chr16    1474
chr10    1434
chr5     1282
chr15    1228
chr20    1079
chr8     1075
chr4      953
chr14     832
chr22     766
chrX      671
chr18     465
chr21     396
chr13     344
Name: enhancer_chrom, dtype: int64
```

[52]:
```
# Filtern zahlen die Promotoren in jedem Chromosom sowie entfernen die␣
 ↪Duplikationen
df_pro_per_chr = df_pair[['promoter_chrom', 'promoter_name']].
 ↪drop_duplicates(ignore_index=True)
print(df_pro_per_chr)
promoters_count_per_chr = df_pro_per_chr['promoter_chrom'].value_counts()
promoters_count_per_chr
```

```
     promoter_chrom                 promoter_name
0              chr1     K562|chr1:6613082-6615021
1              chr1   K562|chr1:10002482-10004387
2              chr1   K562|chr1:10092884-10095822
3              chr1   K562|chr1:16677563-16679703
4              chr1   K562|chr1:17379518-17381119
…               …                            …
7849           chrX   K562|chrX:46404369-46405910
7850           chr4  K562|chr4:146018094-146021046
7851           chr8   K562|chr8:82755000-82755183
```

```
7852           chr5  K562|chr5:127417320-127420890
7853          chr17     K562|chr17:5389709-5390902

[7854 rows x 2 columns]
```

[52]:
```
chr1     832
chr19    643
chr2     500
chr17    491
chr11    454
chr6     453
chr3     435
chr12    423
chr7     394
chr16    366
chr10    314
chr5     310
chr9     307
chr15    258
chr4     256
chr8     249
chr14    241
chrX     235
chr20    223
chr22    173
chr13    120
chr18    100
chr21     77
Name: promoter_chrom, dtype: int64
```

[53]:
```
# Erstellen ein Data Frame, das die Anzahle der Enhancern und Promotoren in den␣
 ↪Chromosomen enthält
en_pro_count_per_chr = pd.merge(enhancers_count_per_chr,␣
 ↪promoters_count_per_chr,
                                left_index=True, right_index=True, how='inner').
 ↪sort_index()
en_pro_count_per_chr
```

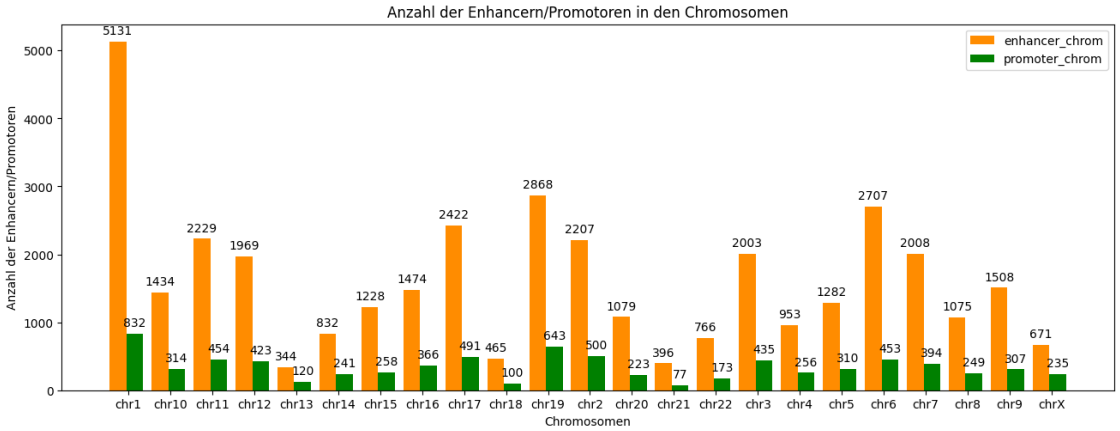
[53]:

	enhancer_chrom	promoter_chrom
chr1	5131	832
chr10	1434	314
chr11	2229	454
chr12	1969	423
chr13	344	120
chr14	832	241
chr15	1228	258
chr16	1474	366

chr17	2422	491
chr18	465	100
chr19	2868	643
chr2	2207	500
chr20	1079	223
chr21	396	77
chr22	766	173
chr3	2003	435
chr4	953	256
chr5	1282	310
chr6	2707	453
chr7	2008	394
chr8	1075	249
chr9	1508	307
chrX	671	235

6.2 Visualisieren die Anzahl der Enhancern und Promotoren an den Chromosomen

[54]:
```
import numpy as np
x = np.arange(len(en_pro_count_per_chr.index))  # the label locations
width = 0.4  # the width of the bars
multiplier = 0

fig, ax = plt.subplots(layout='constrained', figsize=(13,5))

for col in en_pro_count_per_chr.columns:
    offset = width * multiplier
    color = 'darkorange' if col == 'enhancer_chrom' else 'green'
    rects = ax.bar(x + offset, en_pro_count_per_chr[col], width, label=col,␣
 ↪color = color)
    ax.bar_label(rects, padding=4)
    multiplier += 1

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Chromosomen')
ax.set_ylabel('Anzahl der Enhancern/Promotoren')
ax.set_title('Anzahl der Enhancern/Promotoren in den Chromosomen')
ax.set_xticks(x + 0.25, en_pro_count_per_chr.index)

ax.legend(loc='upper right', ncols=1)

plt.show()
```


Anzahl der Enhancern/Promotoren in den Chromosomen
enhancer_chrom
promoter_chrom
Anzahl der Enhancern/Promotoren
Chromosomen
0
1000
2000
3000
4000
5000
chr1 5131 832
chr10 1434 314
chr11 2229 454
chr12 1969 423
chr13 344 120
chr14 832 241
chr15 1228 258
chr16 1474 366
chr17 2422 491
chr18 465 100
chr19 2868 643
chr2 2207 500
chr20 1079 223
chr21 396 77
chr22 766 173
chr3 2003 435
chr4 953 256
chr5 1282 310
chr6 2707 453
chr7 2008 394
chr8 1075 249
chr9 1508 307
chrX 671 235

[ ]: