instance-based-learning

February 8, 2024

1 INSTANCE BASED LEARNING

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.
 ↪data"

headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',␣
 ↪'Class']
dataset = pd.read_csv(path, names = headernames)
dataset.head()

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.40)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 8)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
```

```
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[21  0  0]
 [ 0 23  1]
 [ 0  1 14]]
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        21
Iris-versicolor       0.96      0.96      0.96        24
 Iris-virginica       0.93      0.93      0.93        15

       accuracy                           0.97        60
      macro avg       0.96      0.96      0.96        60
   weighted avg       0.97      0.97      0.97        60

Accuracy: 0.9666666666666667
```

[12]:
```
import numpy as np
import pandas as pd

path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.
 ↪data"

headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',␣
 ↪'Class']

data = pd.read_csv(path, names=headernames)
array = data.values
X = array[:, :2]
y = array[:, 2]

from sklearn.neighbors import KNeighborsRegressor
knnr = KNeighborsRegressor(n_neighbors=10)
knnr.fit(X, y)

print("The MSE is:", format(np.power(y - knnr.predict(X), 2).mean()))
```

```
The MSE is: 0.1706753333333333
```

[ ]:
```

```

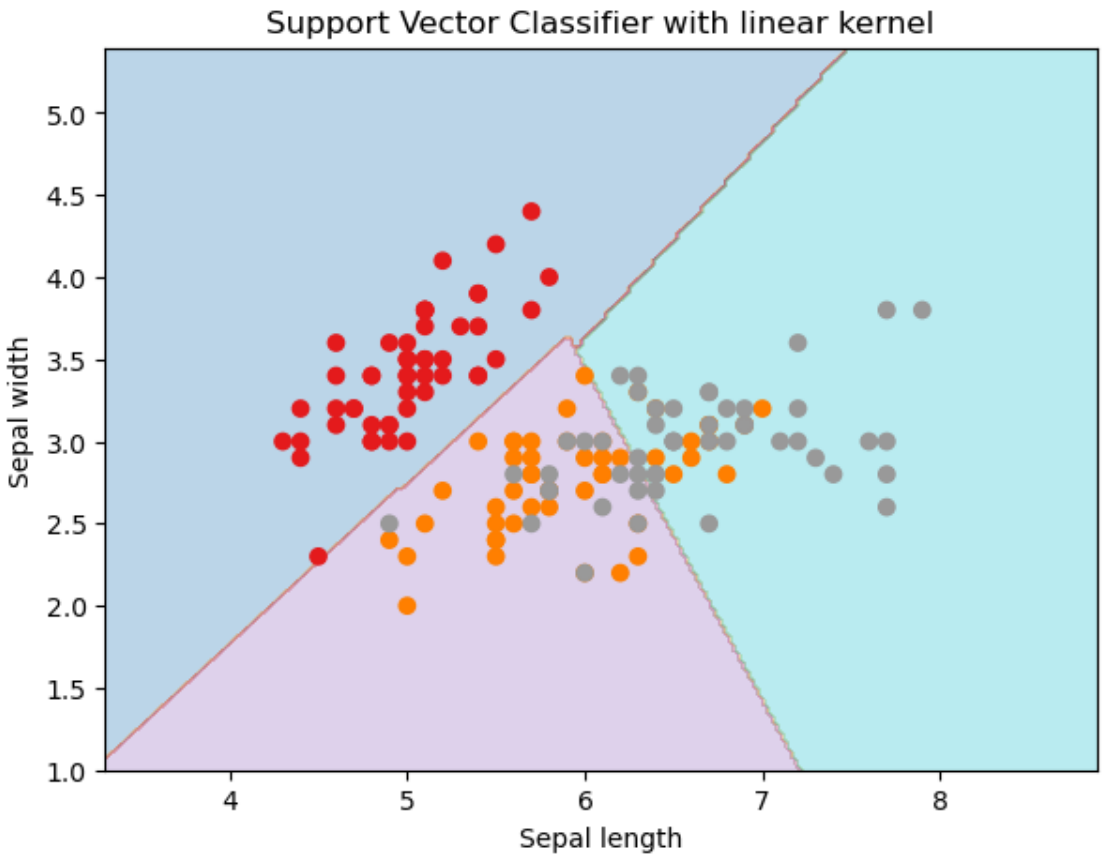
support-vector-machine-svm

February 8, 2024

1 SUPPORT VECTOR MACHINE

[3]:
```
import pandas as pd
import numpy as np
from sklearn import svm, datasets
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min) / 100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
X_plot = np.c_[xx.ravel(), yy.ravel()]

C = 1.0

svc_classifier = svm.SVC(kernel='linear', C=C).fit(X, y)
Z = svc_classifier.predict(X_plot)
Z = Z.reshape(xx.shape)
plt.figure(figsize=(15, 5))
plt.subplot(121)
plt.contourf(xx, yy, Z, cmap=plt.cm.tab10, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Classifier with linear kernel')
```

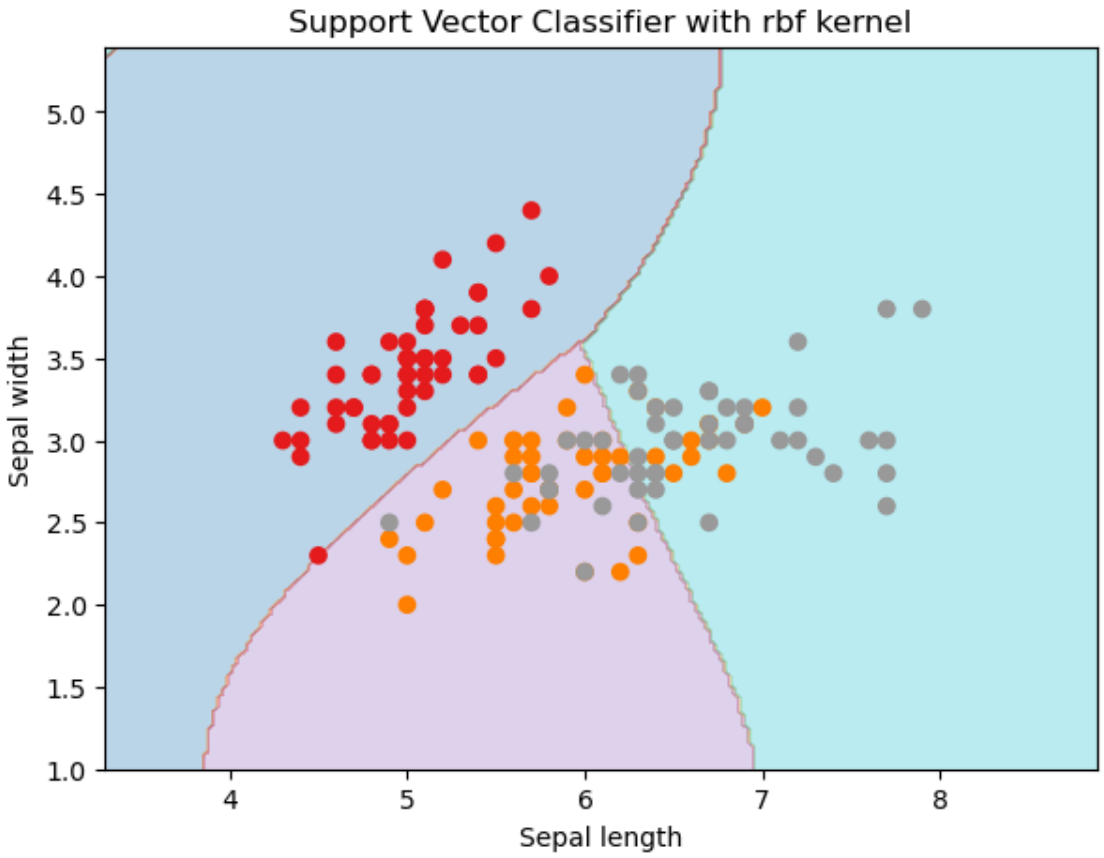
[3]:
```
Text(0.5, 1.0, 'Support Vector Classifier with linear kernel')
```



[4]:
```
import pandas as pd
import numpy as np
from sklearn import svm, datasets
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min) / 100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
X_plot = np.c_[xx.ravel(), yy.ravel()]

C = 1.0

svc_classifier = svm.SVC(kernel='rbf', gamma='auto', C=C).fit(X, y)
Z = svc_classifier.predict(X_plot)
Z = Z.reshape(xx.shape)
```

```
plt.figure(figsize=(15, 5))
plt.subplot(121)
plt.contourf(xx, yy, Z, cmap=plt.cm.tab10, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Classifier with rbf kernel')
```

[4]: Text(0.5, 1.0, 'Support Vector Classifier with rbf kernel')



[ ]:

hidden-markov-model

February 8, 2024

1 HIDDEN MARKOV MODEL

[10]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm
from matplotlib import pyplot as plt

from sklearn.model_selection import GroupShuffleSplit
from hmmlearn import hmm
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

dataset = pd.read_csv("ner_dataset.csv", encoding='latin1')
dataset = dataset.fillna(method="ffill")
dataset = dataset.rename(columns={'Sentence #': 'sentence'})
dataset.head(5)
```

[10]:

	sentence	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	Sentence: 1	of	IN	O
2	Sentence: 1	demonstrators	NNS	O
3	Sentence: 1	have	VBP	O
4	Sentence: 1	marched	VBN	O

[12]:
```
tags = list(set(dataset.POS.values))
words = list(set(dataset.Word.values))
len(tags), len(words)
```

[12]: (42, 35177)

[13]:
```
y = dataset.POS
X = dataset.drop('POS', axis=1)

groupshufflesplit = GroupShuffleSplit(n_splits=2, test_size=.33,␣
 ↪random_state=42)
```

```
ix_train, ix_test = next(groupshufflesplit.split(X, y,␣
 ↪groups=dataset['sentence']))

# Use the correct DataFrame name here
dataset_train = dataset.loc[ix_train]
dataset_test = dataset.loc[ix_test]

dataset_train
```

[13]:

	sentence	Word	POS	Tag
24	Sentence: 2	Families	NNS	O
25	Sentence: 2	of	IN	O
26	Sentence: 2	soldiers	NNS	O
27	Sentence: 2	killed	VBN	O
28	Sentence: 2	in	IN	O
…	…	…	…	..
1048570	Sentence: 47959	they	PRP	O
1048571	Sentence: 47959	responded	VBD	O
1048572	Sentence: 47959	to	TO	O
1048573	Sentence: 47959	the	DT	O
1048574	Sentence: 47959	attack	NN	O

[702936 rows x 4 columns]

[14]:
```
tags = list(set(dataset_train.POS.values))
words = list(set(dataset_train.Word.values))
len(tags), len(words)
```

[14]: (42, 29586)

[15]:
```
dataframe_update = dataset_train.sample(frac=.15, replace=False,␣
 ↪random_state=42)
dataframe_update.Word = 'UNKNOWN'
dataset_train.update(dataframe_update)
words = list(set(dataset_train.Word.values))
# Convert words and tags into numbers
word2id = {w: i for i, w in enumerate(words)}
tag2id = {t: i for i, t in enumerate(tags)}
id2tag = {i: t for i, t in enumerate(tags)}
len(tags), len(words)
```

[15]: (42, 27553)

[18]:
```
tags_count = dict(dataset_train.POS.value_counts())
tags_to_word_count = (
    dataset_train.groupby(['POS'])
    .apply(lambda grp: grp.groupby('Word')['POS'].count().to_dict())
```

```
    .to_dict()
)
init_tags_count = dict(dataset_train.groupby('sentence').first().POS.
 ↪value_counts())

tags_to_next_tags_count = np.zeros((len(tags), len(tags)), dtype=int)
sentences = list(dataset_train.sentence)
pos = list(dataset_train.POS)
for i in range(len(sentences)):
    if (i > 0) and (sentences[i] == sentences[i - 1]):
        prevtagid = tag2id[pos[i - 1]]
        nexttagid = tag2id[pos[i]]
        tags_to_next_tags_count[prevtagid][nexttagid] += 1

my_start_prob = np.zeros((len(tags),))
my_transmat = np.zeros((len(tags), len(tags)))
my_emission_prob = np.zeros((len(tags), len(words)))
num_sentences = sum(init_tags_count.values())
sum_tags_to_next_tags = np.sum(tags_to_next_tags_count, axis=1)

for tag, tagid in tag2id.items():
    floatCountTag = float(tags_count.get(tag, 0))  # Fixed typo here
    my_start_prob[tagid] = init_tags_count.get(tag, 0) / num_sentences
    for word, wordid in word2id.items():
        my_emission_prob[tagid][wordid] = tags_to_word_count.get(tag, {}).
 ↪get(word, 0) / floatCountTag

    for tag2, tagid2 in tag2id.items():
        my_transmat[tagid][tagid2] = tags_to_next_tags_count[tagid][tagid2] /␣
 ↪sum_tags_to_next_tags[tagid]
```

[19]:
```
model = hmm.MultinomialHMM(n_components=len(tags), algorithm='viterbi',␣
 ↪random_state=42
)
model.startprob_ = my_start_prob
model.transmat_ = my_transmat
model.emissionprob_ = my_emission_prob
```

```
MultinomialHMM has undergone major changes. The previous version was
implementing a CategoricalHMM (a special case of MultinomialHMM). This new
implementation follows the standard definition for a Multinomial distribution
(e.g. as in https://en.wikipedia.org/wiki/Multinomial_distribution). See these
issues for details:
https://github.com/hmmlearn/hmmlearn/issues/335
https://github.com/hmmlearn/hmmlearn/issues/340
```

[30]:
```
import pandas as pd
from hmmlearn import hmm

# Assuming dataset_test is a pandas DataFrame
dataset_test.loc[~dataset_test['Word'].isin(words), 'Word'] = 'UNKNOWN'
test_word = list(dataset_test.Word)
samples_of = []
for i, val in enumerate(test_word):
    samples_of.append([word2id[val]])

# Using pandas for sentence length calculation
lengths = []
count = 0
sentences = list(dataset_test.sentence)
for i in range(len(sentences)):
    if (i > 0) and (sentences[i] == sentences[i - 1]):
        count += 1
    elif i > 0:
        lengths.append(count)
        count = 1
    else:
        count = 1

# Initialize the HMM model with n_trials set to 1
model = hmm.MultinomialHMM(n_components=num_states, n_iter=num_iterations,␣
 ↪n_trials=1)

# Train your model if needed
# model.fit(training_data)

# Predict using the trained model
predict_pos = model.predict(samples_of, lengths)
predict_pos
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[30], line 25
     22         count = 1
     24 # Initialize the HMM model with n_trials set to 1
---> 25 model = hmm.MultinomialHMM(n_components=num_states,␣
 ↪n_iter=num_iterations, n_trials=1)
     27 # Train your model if needed
     28 # model.fit(training_data)
     29
     30 # Predict using the trained model
     31 predict_pos = model.predict(samples_of, lengths)
```

```
NameError: name 'num_states' is not defined
```

[1]:
```
nbconvert --allow-chromium-download Hidden Markov Model.ipynb
```

```
  Cell In[1], line 1
    nbconvert --allow-chromium-download Hidden Markov Model.ipynb
                                        ^
SyntaxError: invalid syntax
```

[ ]:
```

```