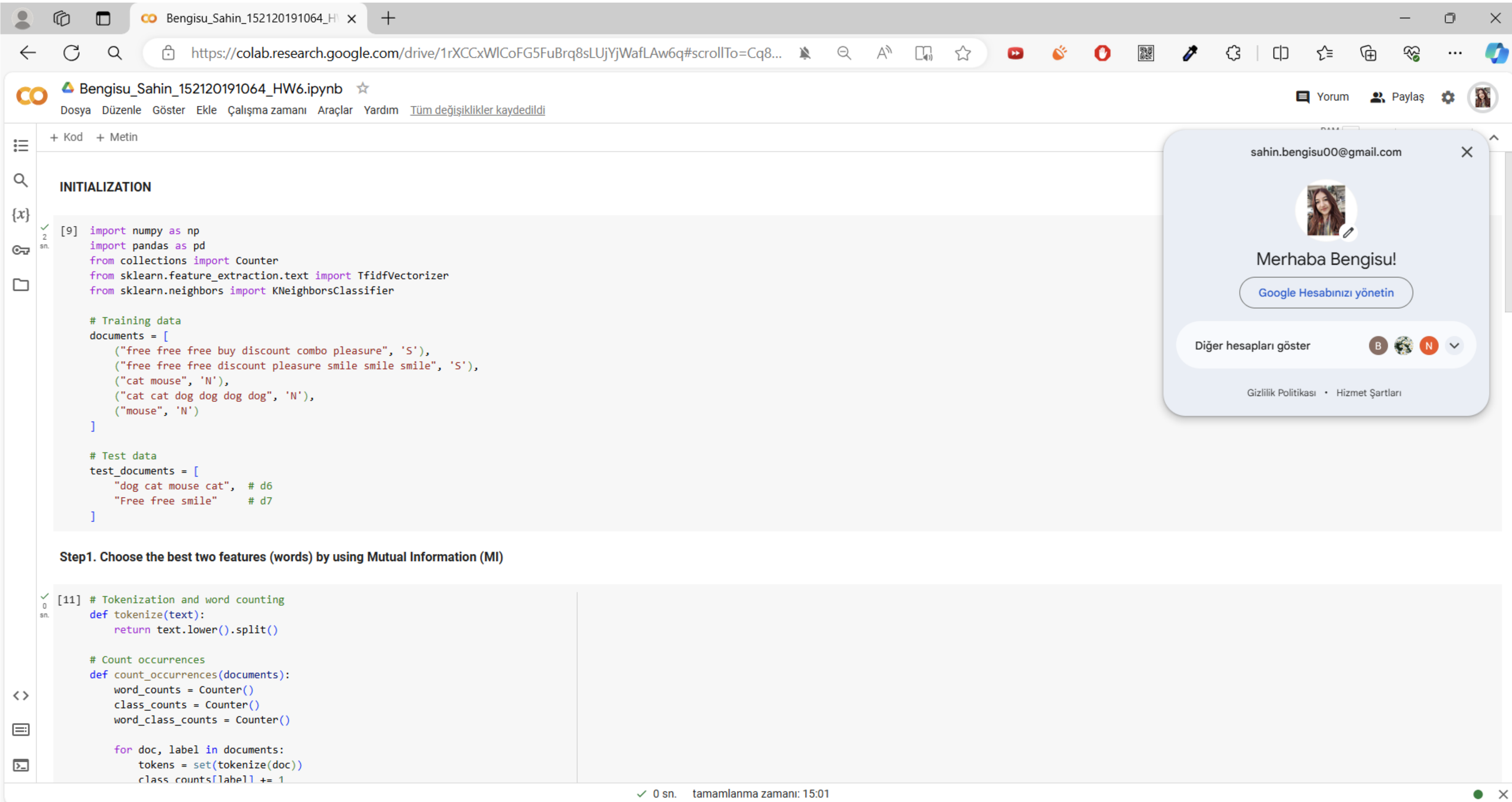
Bengisu Şahin

152120191064

HW6 Report

1.



2.

This code implements a text classification system using Mutual Information (MI) and a K-Nearest Neighbors (KNN) classifier. It starts with a dataset of labeled training documents and test documents. The training documents are tokenized and word occurrences are counted. The `tokenize` function converts the text to lowercase and splits it by spaces, while the count_occurrences function counts the occurrences of each word in the documents, the class labels, and the occurrences of each word within each class.Next, Mutual Information (MI) values are calculated for each word to determine how informative each word is for distinguishing between classes. The `compute_mi` function uses the probabilities of words and classes to compute these MI values. The top 2 words with the highest MI scores are selected as features, representing the most informative words for classification. For instance, the selected features might be "free" and "discount".Using `TfidfVectorizer`, the selected words are used to create a TF-IDF matrix for the documents. This matrix represents how strongly each document is associated with the selected words. For the training data, a 5x2 matrix is obtained. The KNN classifier is then trained using this TF-IDF matrix and the labels from the training data. The same vectorizer is used to create TF-IDF vectors for the test documents, and the KNN classifier predicts the class labels for these test documents based on their TF-IDF vectors.Finally, for the documents "dog cat mouse cat" (d6) and "Free free smile" (d7), the classifier predicts the classes, for example, N and S, respectively. This system demonstrates the process of text classification using feature selection based on MI, vectorization with TF-IDF, and classification with a KNN classifier. Each step is designed to utilize the statistical properties of the text data to achieve accurate classification.

Step1. Choose the best two features (words) by using Mutual Information (MI)

```python
# Tokenization and word counting
def tokenize(text):
    return text.lower().split()

# Count occurrences
def count_occurrences(documents):
    word_counts = Counter()
    class_counts = Counter()
    word_class_counts = Counter()

    for doc, label in documents:
        tokens = set(tokenize(doc))
        class_counts[label] += 1
        for token in tokens:
            word_counts[token] += 1
            word_class_counts[(token, label)] += 1

    return word_counts, class_counts, word_class_counts

# Compute Mutual Information (MI)
def compute_mi(word, word_counts, class_counts, word_class_counts, num_docs):
    mi = 0.0
    for cls in class_counts:
        p_w_c = word_class_counts[(word, cls)] / num_docs
        p_w = word_counts[word] / num_docs
        p_c = class_counts[cls] / num_docs
        if p_w_c > 0:
            mi += p_w_c * np.log2(p_w_c / (p_w * p_c))
    return mi

# Get word counts and class counts
word_counts, class_counts, word_class_counts = count_occurrences(documents)

# Calculate MI for each word
num_docs = len(documents)
mi_scores = {word: compute_mi(word, word_counts, class_counts, word_class_counts, num_docs) for word in word_cc

# Select top 2 words with highest MI
selected_features = sorted(mi_scores, key=mi_scores.get, reverse=True)[:2]
print(f"Selected features: {selected_features}")
```

```
Selected features: ['discount', 'free']
```

Step2. You are expected to compute the TF*IDF score of selected two features

---

Step 3: Represent Each Document with Selected Features (TF x IDF Values) .

```python
[13] # Vectorize documents
vectorizer = TfidfVectorizer(vocabulary=selected_features)
X_train = vectorizer.fit_transform([doc for doc, _ in documents])
y_train = [label for _, label in documents]

print("TF-IDF matrix for training data:")
print(X_train.toarray())

# Create TF-IDF matrix (5x2)
tfidf_matrix = X_train.toarray()
print("TF-IDF matrix (5x2):")
print(tfidf_matrix)
```

```
TF-IDF matrix for training data:
[[0.31622777 0.9486833 ]
 [0.31622777 0.9486833 ]
 [0.         0.        ]
 [0.         0.        ]
 [0.         0.        ]]
TF-IDF matrix (5x2):
[[0.31622777 0.9486833 ]
 [0.31622777 0.9486833 ]
 [0.         0.        ]
 [0.         0.        ]
 [0.         0.        ]]
```

Step 4 & 5: Calculate TF*IDF Values for Test Documents

```python
[14] X_test = vectorizer.transform(test_documents)

d6_tfidf = X_test[0].toarray()
d7_tfidf = X_test[1].toarray()

print("TF-IDF vector for d6:")
print(d6_tfidf)

print("TF-IDF vector for d7:")
print(d7_tfidf)
```

```
TF-IDF vector for d6:
[[0. 0.]]
TF-IDF vector for d7:
[[0. 1.]]
```

Step6-7. Predict the class label of d6 & d7 by using the KNN algorithm.

```python
[15] # Train KNN classifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(tfidf_matrix, y_train)

# Predict class labels for d6 and d7
d6_prediction = knn.predict(d6_tfidf)
d7_prediction = knn.predict(d7_tfidf)

print(f"Predicted class for d6: {d6_prediction[0]}")
print(f"Predicted class for d7: {d7_prediction[0]}")
```

```
Predicted class for d6: N
Predicted class for d7: S
```