Machine Learning and Natural Language Processing

Contributors:
• 201805017 NESLİHAN ÖZDİL
• 201805032 ARDIL SİLAN AYDIN
• 201805060 ELİF YILMAZ

1. Introduction

This study aims to develop Turkish text classification applications using the TTC4900 news dataset. In the first phase of the study, common feature extraction methods such as Bag of Words (BoW), TF-IDF, and Word2Vec were used to extract meaningful features from text data. These features will be used for model training, and classification will then be performed using machine learning algorithms such as Random Forest, XGBoost, LightGBM, SVM, and Artificial Neural Networks (ANN).
In the second phase, deep learning methods will be employed by integrating models such as CNN, LSTM, BiLSTM, and GRU, and more complex models will be created by combining these models. Finally, transformer-based models like BERT, fine-tuned for Turkish texts, will be used for classification.
The accuracy of each model will be evaluated using metrics such as precision, recall, F1 score, Confusion Matrix, and AUC (ROC curve), and the most successful model will be selected.

2. Dataset Preparation and Cleaning

Loading and Cleaning the Dataset

The dataset used in the study is the 7allV03.csv file. Before it was made suitable for text classification tasks, the dataset underwent various cleaning and preprocessing steps. The dataset was loaded and cleaned using the Pandas library.

1. **Removing Duplicate Data**: Duplicate rows in the dataset were removed using the `drop_duplicates()` function, and the index was reset to allow further processing.
2. **Correcting Abbreviations**: Abbreviations in the dataset were converted to their meaningful full forms. For example, the abbreviation "AA" was changed to "Anadolu Ajansı".
3. **Text Cleaning**: The text data underwent several cleaning steps:
    - **Lowercasing**: All texts were converted to lowercase.
    - **Removing URLs and Email Addresses**: URLs starting with http and www, as well as email addresses, were removed from the text.
    - **Removing Numbers and Punctuation Marks**: Numbers and punctuation marks were removed from the text.
    - **Removing Unnecessary Characters**: Special characters (e.g., quotation marks) were removed.
    - **Removing Emojis and HTML Tags**: Emojis and HTML tags were removed from the text.
    - **Removing Stopwords**: Turkish stopwords were removed from the text.
    - **Removing Rare Words**: Words that appeared only once were removed.

```python
# Load the dataset
file_path = '7allV03.csv'
df = pd.read_csv(file_path)

df = df.drop_duplicates()
df = df.reset_index(drop=True)

# Example abbreviation dictionary for conversion (you should define y
kisaltmalar = {
    "AA":  "Anadolu Ajansı",
    "AAET":  "Avrupa Atom Enerjisi Topluluğu",
    "AB": "Avrupa Birliği",
    "ABB": "Avrupa Birliği Bankası",
    "ABGS": "Avrupa Birliği Genel Sekreterliği",
    "ABKF": "Avrupa Bölgesel Kalkınma Fonu",
    "ABÖ": "Afrika Birliği Örgütü",
    "ACÜ": "Acıbadem Üniversitesi",
    "AÇSAP": "Ana Çocuk Sağlığı ve Aile Planlaması Genel Müdürlüğü",
    "ADKF": "Abu Dabi Kalkınma Fonu",
```

```python
# Function to convert abbreviations in the text
def convert_abbrev_in_text(text):
    tokens = word_tokenize(text)
    tokens = [kisaltmalar[word] if word in kisaltmalar.keys() else word for word in tokens]
    text = ' '.join(tokens)
    return text

# Apply the preprocessing steps
df["Text Cleaned"] = df["text"].apply(convert_abbrev_in_text)

# Convert to lowercase
df['Text Cleaned'] = [token.lower() for token in df['Text Cleaned']]

# Remove URLs
df['Text Cleaned'] = df['Text Cleaned'].replace(r'http\S+', '', regex=True).replace(r'www\S+', '', regex=True)

# Remove numbers
df['Text Cleaned'] = df['Text Cleaned'].apply(lambda x: re.sub('[0-9]+', '', x))

# Remove punctuation
df['Text Cleaned'] = df['Text Cleaned'].apply(lambda x: x.translate(x.maketrans('', '', string.punctuation)))

# Remove specific unwanted characters
df['Text Cleaned'] = df['Text Cleaned'].apply(lambda x: x.replace('"', '').replace("’", '').replace("'", '').replace("”", ''))

# Remove email addresses
df['Text Cleaned'] = df['Text Cleaned'].apply(lambda x: re.sub('\S*@\S*\s?', '', x))

# Remove emojis
df['Text Cleaned'] = df['Text Cleaned'].apply(lambda x: emoji.replace_emoji(x))

# Remove HTML tags
df['Text Cleaned'] = df['Text Cleaned'].apply(lambda x: re.sub('<.*?>', '', x))

# Load stopwords
stop_words = [x.strip() for x in open('stop-words.tr.txt','r', encoding="UTF8").read().split('\n')]

# Remove stopwords
df['Text Cleaned'] = df['Text Cleaned'].apply(lambda text: ' '.join([word for word in text.split() if word.lower() not in stop_words]))

# Remove less frequent words (words that occur only once)
freq = pd.Series(' '.join(df['Text Cleaned']).split()).value_counts()
less_freq = list(freq[freq == 1].index)
df['Text Cleaned'] = df['Text Cleaned'].apply(lambda x: " ".join(x for x in x.split() if x not in less_freq))

# Show the first few rows of the cleaned text
df[['text', 'Text Cleaned']].head()
```

These steps helped eliminate unnecessary and meaningless information from the dataset, ensuring the model worked more efficiently.

Zemberek Stemmer and Stanza Lemmatization

To better understand the meaning of Turkish texts, lemmatization and stemming processes were performed using Zemberek and Stanza tools.

1. **Zemberek Stemming**: Zemberek is a powerful tool that can analyze the morphological features of Turkish. Each word in the texts was analyzed using

Zemberek and reduced to its root form. This process was a critical step, especially in processing the rich morphological structure of Turkish.

2. **Stanza Lemmatization**: Stanza is a deep learning-based modern lemmatization tool. It was used to find the base form of each word corresponding to its meaning. The Turkish model of Stanza was utilized to convert the words in the texts into meaningful stems.

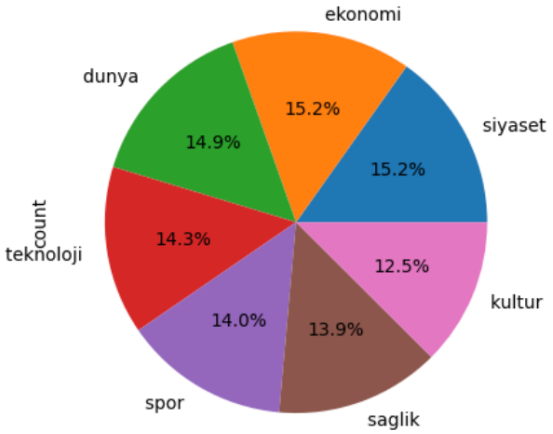
By using these two tools together, the grammatical analysis of Turkish texts was performed, and the morphological analysis of the texts was successfully completed.

```python
# Zemberek JAR dosyasının yolu
ZEMBEREK_PATH = 'zemberek-full.jar'

# JAR dosyasının varlığını kontrol et
if not os.path.exists(ZEMBEREK_PATH):
    raise FileNotFoundError("Zemberek JAR dosyası bulunamadı. Lütfen yolu kontrol edin.")

# JVM'yi başlat
if not jpype.isJVMStarted():
    startJVM(getDefaultJVMPath(), '-ea', f'-Djava.class.path={ZEMBEREK_PATH}')

# Zemberek sınıflarını yükle
TurkishMorphology = JClass('zemberek.morphology.TurkishMorphology')
morphology = TurkishMorphology.createWithDefaults()

# Stanza Türkçe modeli indirme ve başlatma
stanza.download('tr')
nlp = stanza.Pipeline('tr', processors='tokenize,mwt,pos,lemma', use_gpu=True)

# Zemberek ile kök bulma fonksiyonu
def zemberek_root_extraction(text):
    analysis = morphology.analyzeAndDisambiguate(JString(text)).bestAnalysis()
    roots = []
    for a in analysis:
        roots.append(str(a.getStem()))  # Kök değerini listeye ekle
    return ' '.join(roots)

def stanza_lemmatization(text):
    doc = nlp(text)
    lemmas = []
    for sentence in doc.sentences:
        for word in sentence.words:
            # Eğer lemma None ise orijinal kelimeyi kullan
            lemmas.append(word.lemma if word.lemma is not None else word.text)
    return ' '.join(lemmas)
```

Results

After the data cleaning and preprocessing steps, the distribution of texts across categories is summarized as follows: Economy: 690 examples, Politics: 690 examples, World: 676 examples, Technology: 649 examples, Sports: 635 examples, Health: 631 examples, Culture: 567 examples. This cleaning process was carried out to achieve more effective results in tasks such as text mining and text classification.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4540 entries, 0 to 4539
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   category      4540 non-null   object
 1   text          4540 non-null   object
 2   Text Cleaned  4540 non-null   object
dtypes: object(3)
memory usage: 106.5+ KB
```



	category	text	Text Cleaned	Zemberek Roots	Stanza Lemmas
0	siyaset	3 milyon ile ön seçim vaadi mhp nin 10 olağan...	ön seçim vaadi mhp nin olağan büyük kurultayı ...	ön seçim vaadi mhp nin olağan büyük kurulta nd...	ön seçim vaad mhp nin olağan büyük kurulta nda...
1	siyaset	mesut_yılmaz yüce_divan da ceza alabilirdi pr...	mesutyılmaz yücedivan ceza alabilirdi prof dr ...	mesutyılmaz yücedivan ceza al prof dr sacit ad...	mesutyılmaz yücedivan ceza al prof dr sacit ad...
2	siyaset	disko lar kaldırılıyor başbakan_yardımcısı ar...	disko lar kaldırılıyor başbakanyardımcısı arın...	disko lar kal başbakanyar arınç disko tabir et...	disko lar kal başbakanyar arınç disko tabir et...
3	siyaset	sarıgül anayasa_mahkemesi ne gidiyor mustafa_...	sarıgül anayasamahkemesi gidiyor mustafasarıgü...	sarıgül anayasamahkesi git mustafasarıgül ilçe...	sarıgül anayasamahkesi git mustafasarıgül ilçe...
4	siyaset	erdoğan idamın bir haklılık sebebi var demek ...	erdoğan idamın haklılık sebebi var yeri geldiğ...	erdoğan idam haklılık sebep var yer gel zaman ...	erdoğan idam hak sebep var yer gel zaman idam ...
5	siyaset	hüseyin_çelik bunu kim yaparsa pahalıya ödeti...	hüseyinçelik bunu yaparsa pahalıya ödetiriz ak...	hüseyinçelik bu yap pahalı öde akparti genelba...	hüseyinçelik bu yap pahalı öde akparti genelba...
6	siyaset	yılmaz_özdil e bira cevabı ak_parti milletvek...	yılmazözdil bira cevabı akparti milletvekili ş...	yılmazözdil bira cevap akparti milletvekili şa...	yılmazözdil bira cevap akparti milletvekili şa...
7	siyaset	bakanlıklar lale_devri nde mhp ankara milletv...	bakanlıklar laledevri nde mhp ankara milletvek...	bakanlık laledevri nde mhp ankara milletvekili...	bakanlık laledevri n mhp ankara milletvekili ö...
8	siyaset	iktidarın gerçek niyeti ortaya çıktı chp gene...	iktidarın gerçek niyeti ortaya çıktı chp genel...	iktidar gerçek niyet orta çık chp genelbaşkany...	iktidar gerçek niyet orta çık chp genelbaşkany...

3. Machine Learning Models and Parameters

The machine learning algorithms and models used in the study are as follows:

Models

1. **Random Forest**: Classification is performed with 100 decision trees. Parameters: n_estimators=100, random_state=42

2. **SVM**: Classification is performed using a linear kernel. Parameters: kernel='linear', random_state=42, probability=True
3. **XGBoost**: Classification is performed using fast gradient boosting. Parameters: random_state=42, use_label_encoder=False
4. **LightGBM**: A fast and efficient gradient boosting model is used. Parameters: random_state=42, verbose=-1

Feature Extraction Methods

- BoW (Bag of Words), TF-IDF, and Word2Vec methods were used to convert the texts into numerical data. These methods enable each word to be transformed into a numerical representation and can be used for text classification tasks.

Evaluation Methods
The performance of each model was evaluated using metrics such as accuracy, precision, recall, F1 score, and AUC. The performance was visualized using the ROC curve and confusion matrix.

Model	BoW F1	BoW Accuracy	TF-IDF F1	TF-IDF Accuracy	Word2Vec F1	Word2Vec Accuracy
Random Forest	0.8638	0.8645	0.8569	0.8579	0.7897	0.7896
SVM	0.8056	0.8062	0.8946	0.8943	0.8160	0.8150
XGBoost	0.8877	0.8877	0.8714	0.8711	0.8062	0.8062
LightGBM	0.8833	0.8833	0.8897	0.8899	0.8051	0.8051

Best Model: SVM - TF-IDF

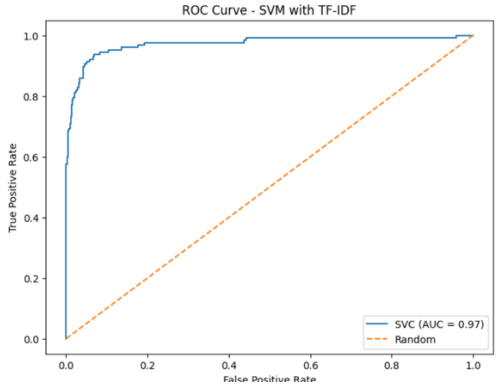
- **F1 Score: 0.8946**
- **Accuracy Score: 0.8943**

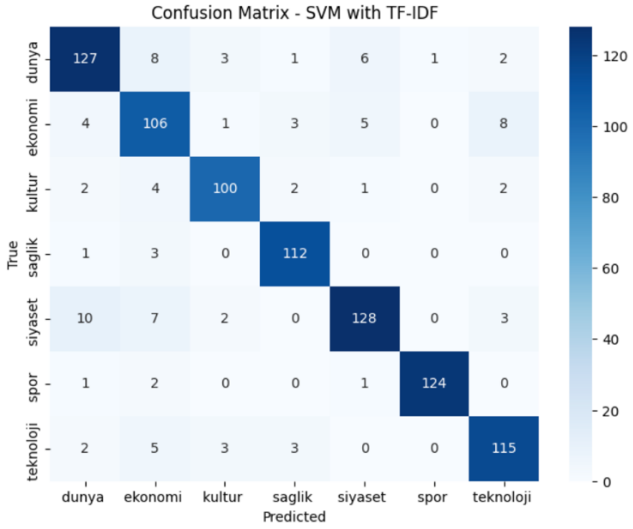
Worst Model: Random Forest - Word2Vec

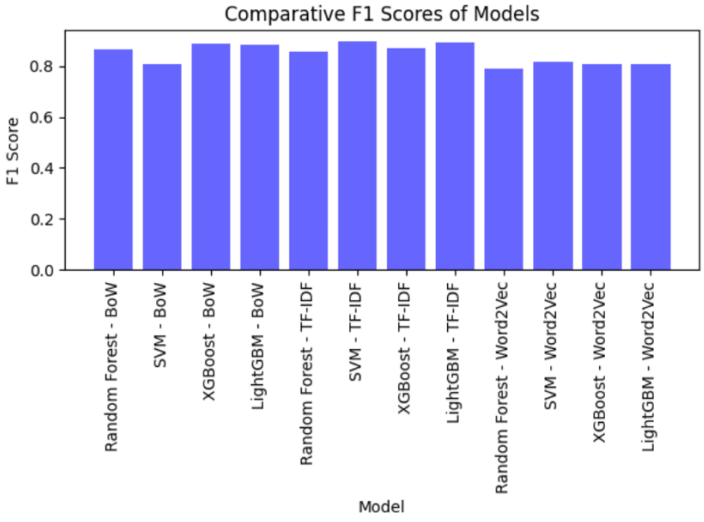
- **F1 Score: 0.7897**
- **Accuracy: 0.7896**

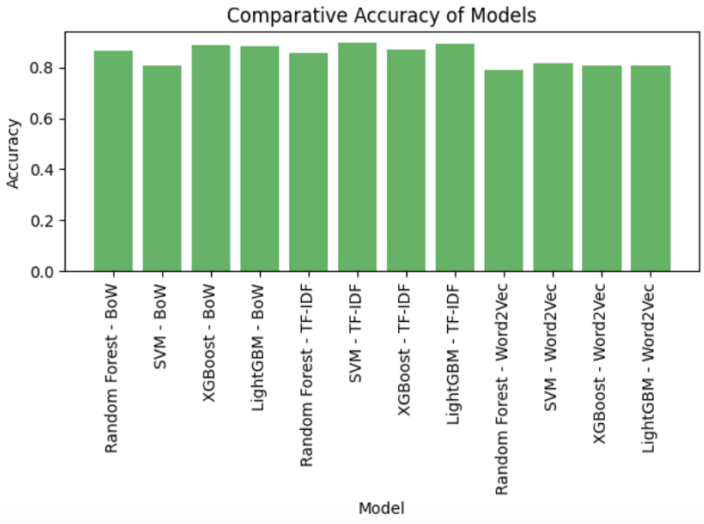
Classification Report for SVM with TF-IDF features:

	precision	recall	f1-score	support
dunya	0.86	0.86	0.86	148
ekonomi	0.79	0.83	0.81	127
kultur	0.92	0.90	0.91	111
saglik	0.93	0.97	0.95	116
siyaset	0.91	0.85	0.88	150
spor	0.99	0.97	0.98	128
teknoloji	0.88	0.90	0.89	128
accuracy			0.89	908
macro avg	0.90	0.90	0.90	908
weighted avg	0.90	0.89	0.89	908









4. Deep Learning Models

ANN

Artificial Neural Network (ANN) was used for Turkish text classification. The model was trained using features obtained from the Bag of Words (BoW), TF-IDF, and Word2Vec methods during the feature extraction phase. Additionally, SMOTE (Synthetic Minority Over-sampling Technique) was applied to address class imbalance.

Model Design:
The model utilizes the ReLU activation function and Dropout to prevent overfitting. The output layer uses the Softmax activation function for multi-class classification with 7 classes. The model was trained using the Adam optimizer and sparse categorical cross-entropy loss function.

```
def create_ann_model(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_dim=input_dim),
        BatchNormalization(),
        Dropout(0.5),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(7, activation='softmax')  # 7 kategori için çıkış katmanı
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model
```

Evaluation:
The model was evaluated using metrics such as accuracy, precision, recall, F1 score, and AUC. Additionally, the ROC curve and confusion matrix were visualized. SMOTE was applied during training to resolve the data imbalance.

Results and Visualization:
The model performance was compared for each feature extraction method, and metrics such as F1 score and accuracy were presented.

	Model	F1 Score	Accuracy
1	ANN-BoW	0.9106	0.9108
2	ANN-TF-IDF	0.9035	0.9031
3	ANN-Word2Vec	0.8058	0.8051

Best Model ANN-BoW:

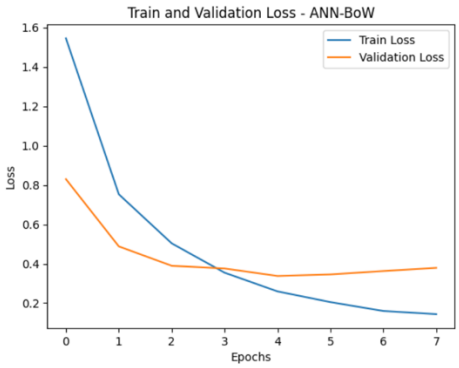
- **F1 Score: 0.9106**
- **Accuracy: 0.9108**

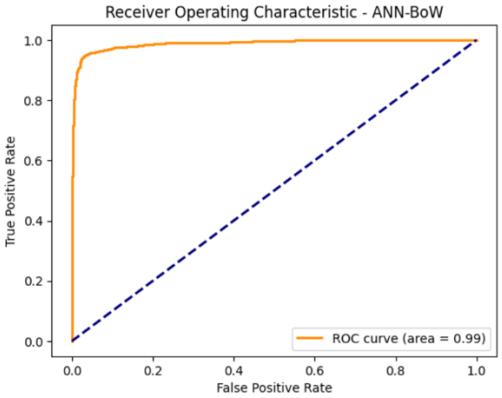
Worst Model ANN-Word2Vec:

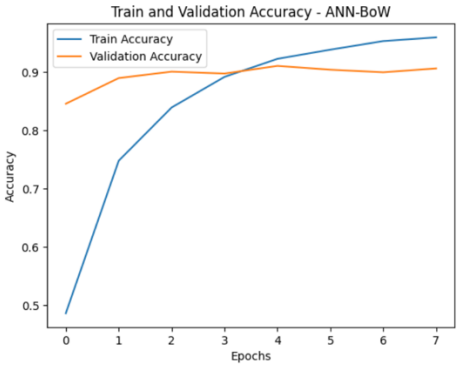
- **F1 Score: 0.8058**
- **Accuracy: 0.8051**

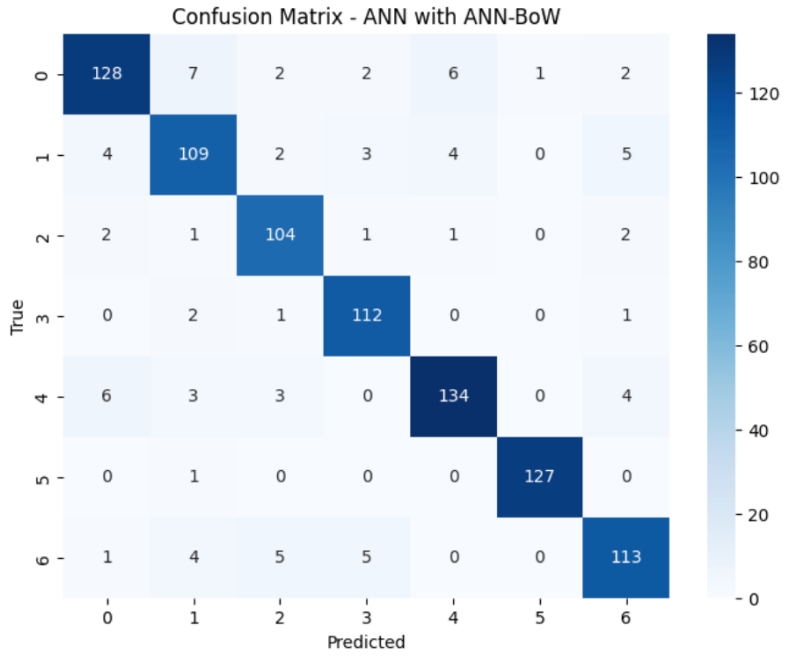
```
Epoch 8/50
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9640 - loss: 0.1
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Classification Report for ANN with ANN-BoW features:
              precision    recall  f1-score   support

           0       0.91      0.86      0.89       148
           1       0.86      0.86      0.86       127
           2       0.89      0.94      0.91       111
           3       0.91      0.97      0.94       116
           4       0.92      0.89      0.91       150
...
    accuracy                           0.91       908
   macro avg       0.91      0.91      0.91       908
weighted avg       0.91      0.91      0.91       908
```

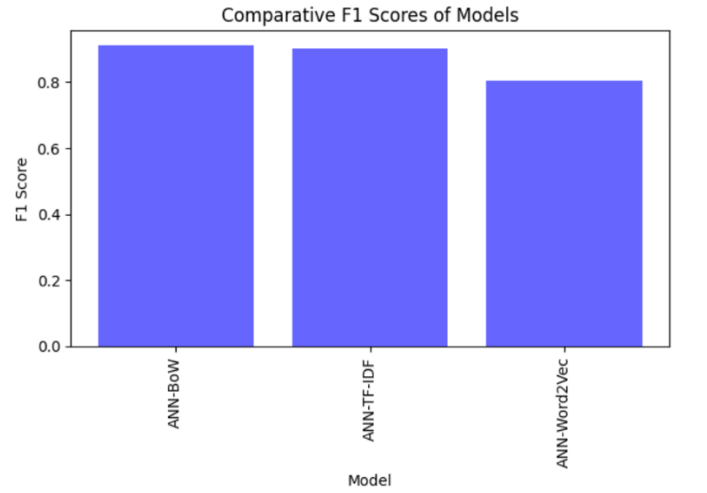
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output s

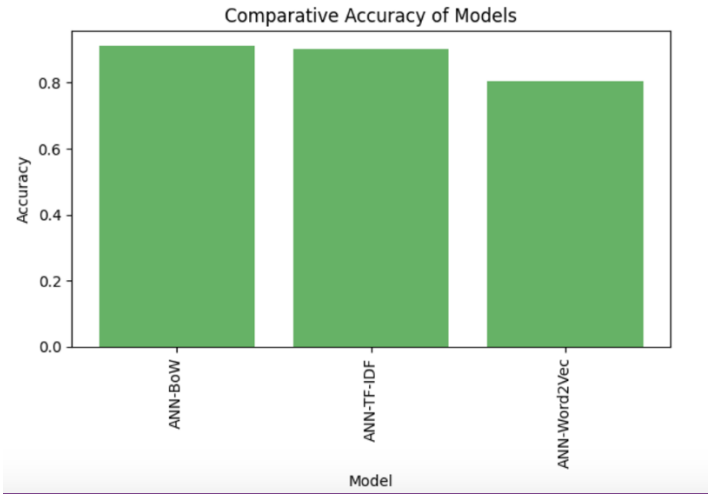












CNN, LSTM, BiLSTM, and GRU Models

For text classification, a total of 11 different models were created using combinations of CNN, LSTM, BiLSTM, and/or GRU. In each model, various feature extraction methods and embedding strategies were applied. The training and testing results of these models were evaluated using metrics such as accuracy, precision, recall, F1 score, ROC curve, and confusion matrix.

Models and Parameters

The models used in this study are grouped into two main categories: Random Initialization and Word2Vec Embedding.

1. **Random Initialization**:
   In this method, random initial embedding values are assigned to words. While this approach allows for faster training of the model, it may be limited in learning the deep linguistic context between words.
2. **Word2Vec Embedding**:
   In this method, the pre-trained Word2Vec model is used to provide meaningful, context-based embedding values for words. This model learns the linguistic context between words more effectively, producing stronger and more meaningful results.

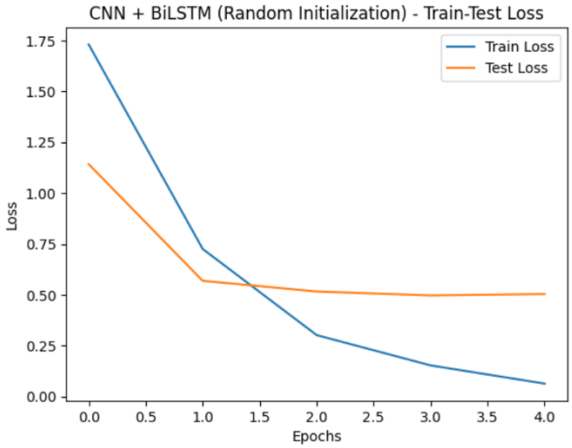
Model	Random - Epoch	Random - ACC	Word2Vec - Epoch	Word2Vec - ACC
CNN + LSTM	5	0.8491	5	0.7676
CNN + BiLSTM	5	0.8711	15	0.8469
CNN + GRU	5	0.7489	40	0.6839
LSTM + BiLSTM	10	0.8150	8	0.8139
LSTM + GRU	23	0.7026	35	0.6641
BiLSTM + GRU	19	0.6839	20	0.4868
CNN + LSTM + BiLSTM	8	0.8425	13	0.8205
CNN + LSTM + GRU	11	0.8403	20	0.8458
CNN + BiLSTM + GRU	17	0.8293	20	0.8392
LSTM + BiLSTM + GRU	23	0.7291	5	0.5661
CNN + LSTM + BiLSTM + GRU	25	0.8326	28	0.8425

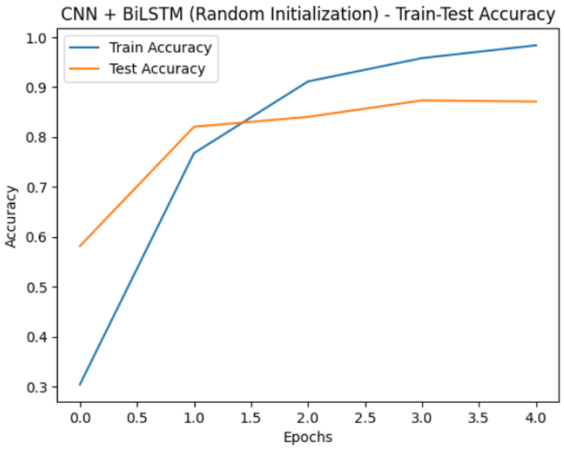
Best Model:

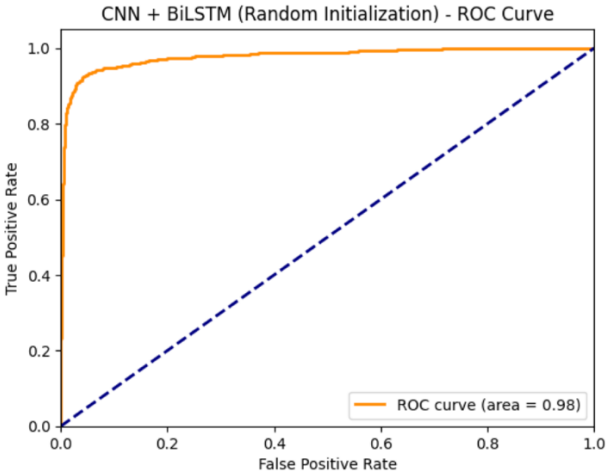
- **Model**: CNN + BiLSTM
- **Epoch (Random Initialization)**: 5
- **Accuracy (Random Initialization)**: 0.8711
- **Epoch (Word2Vec)**: 15
- **Accuracy (Word2Vec)**: 0.8469

Worst Model:

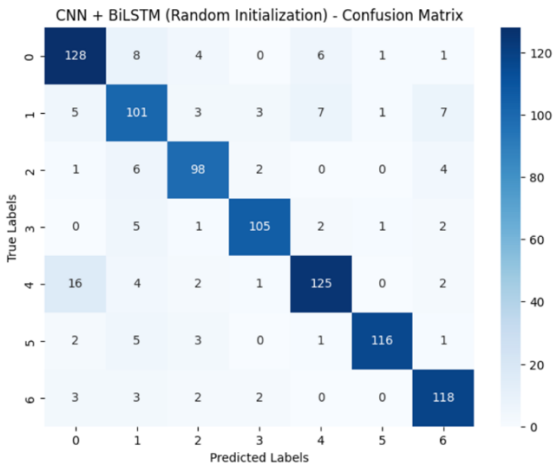
- **Model**: BiLSTM + GRU
- **Epoch (Random Initialization)**: 19
- **Accuracy (Random Initialization)**: 0.6839
- **Epoch (Word2Vec)**: 20
- **Accuracy (Word2Vec)**: 0.4868

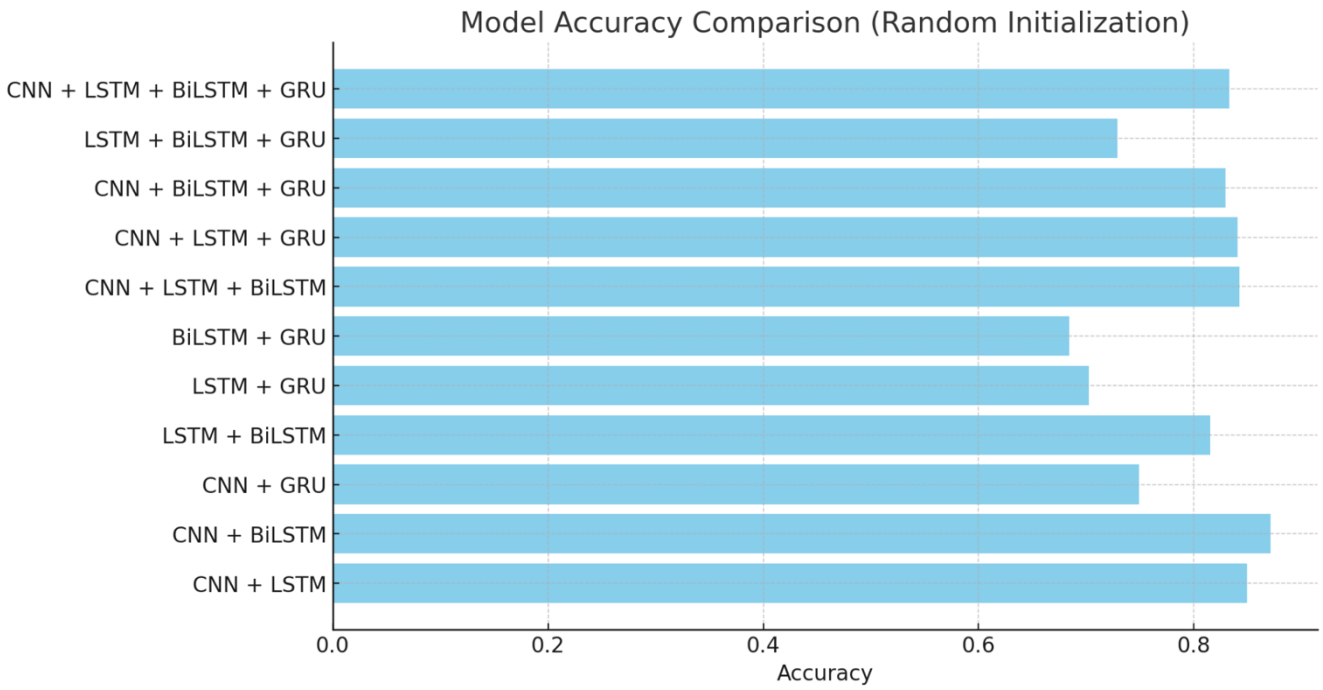


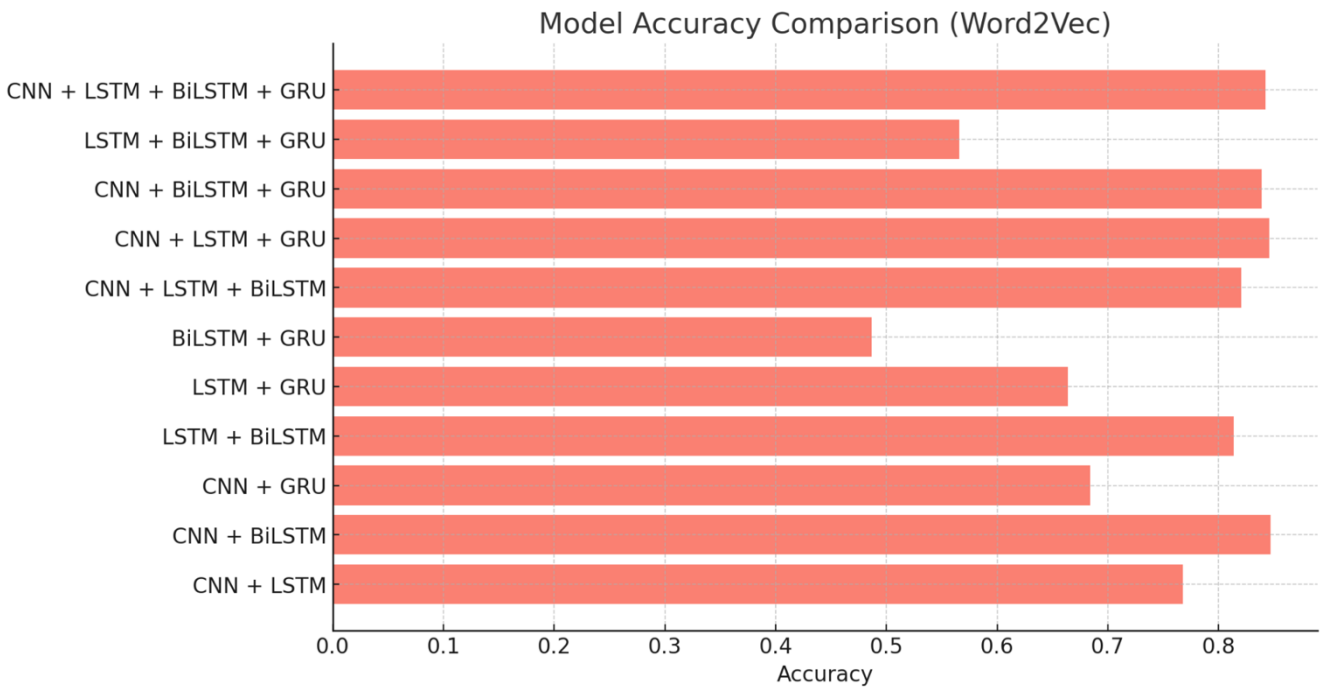





```
Epoch 5/5
57/57 ━━━━━━━━━━━━━━━━━━━━ 10s 174ms/step - accuracy: 0.9854 - l
Training CNN + BiLSTM (Random Initialization)...
29/29 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step
CNN + BiLSTM (Random Initialization) - Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.86      0.84       148
           1       0.77      0.80      0.78       127
           2       0.87      0.88      0.88       111
           3       0.93      0.91      0.92       116
           4       0.89      0.83      0.86       150
           5       0.97      0.91      0.94       128
           6       0.87      0.92      0.90       128

    accuracy                           0.87       908
   macro avg       0.87      0.87      0.87       908
weighted avg       0.87      0.87      0.87       908

...
Accuracy: 0.8711
Precision: 0.8734
Recall: 0.8711
F1-Score: 0.8717
```







5. Transformer Model Results: BERT for Turkish Text Classification

In this study, two BERT-based models were used to solve the Turkish text classification problem. The first model is the multilingual BERT model, bert-base-multilingual-uncased, and the second one is the savasy/bert-turkish-text-classification model, which has been fine-tuned specifically for Turkish texts.

K-Fold Cross-Validation Results

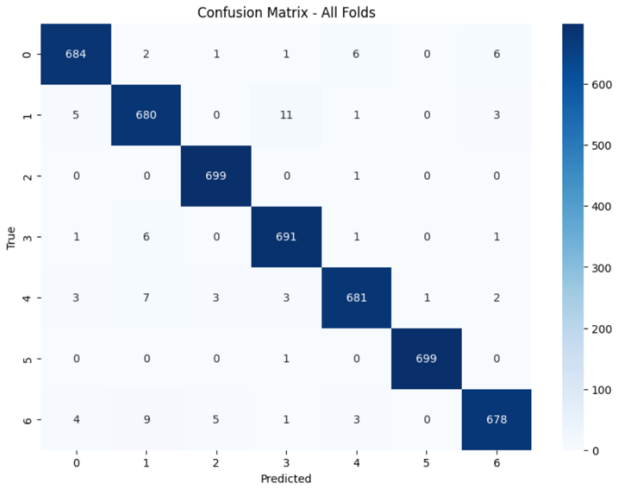
- **savasy/bert-turkish-text-classification**: The average accuracy obtained with K-Fold cross-validation is 0.98204.
- **bert-base-multilingual-uncased**: The average accuracy obtained with K-Fold cross-validation is 0.95408.

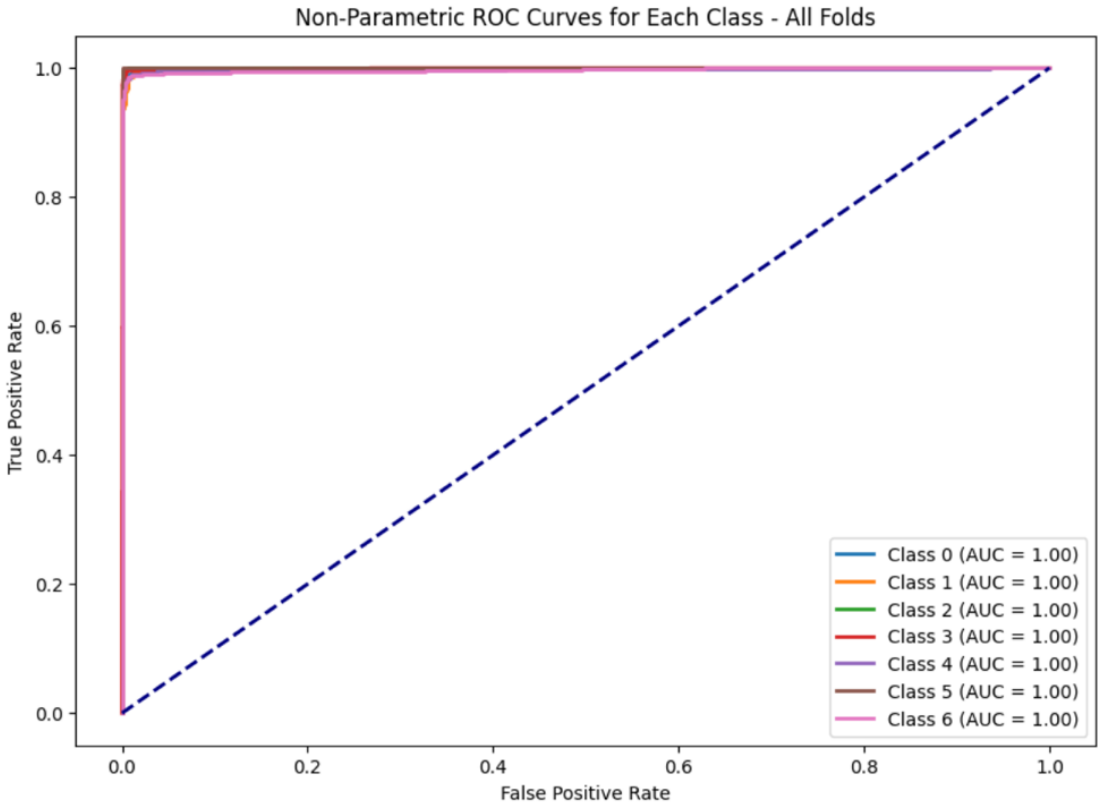
Direct Model Usage Results

- **savasy/bert-turkish-text-classification**: The accuracy obtained with direct training is 0.9551.
- **bert-base-multilingual-uncased**: The accuracy obtained with direct training is 0.9010.

K-fold savasy/bert-turkish-text-classification:

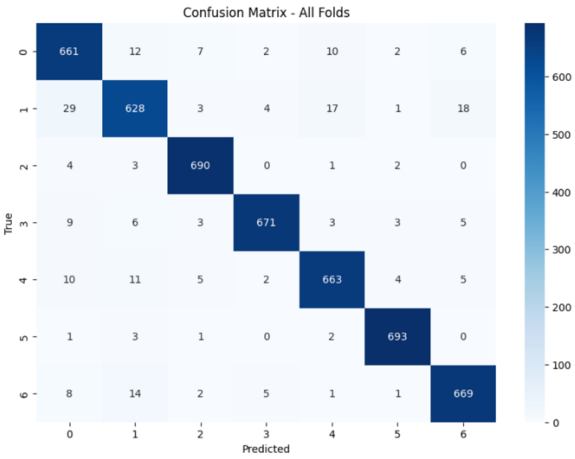
```
Mean Accuracy: 0.9820408163265306
Mean Precision: 0.9822349660073384
Mean Recall: 0.9820408163265306
Mean F1-Score: 0.9820229510011232

Final Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       700
           1       0.97      0.97      0.97       700
           2       0.99      1.00      0.99       700
           3       0.98      0.99      0.98       700
           4       0.98      0.97      0.98       700
           5       1.00      1.00      1.00       700
           6       0.98      0.97      0.98       700

    accuracy                           0.98      4900
   macro avg       0.98      0.98      0.98      4900
weighted avg       0.98      0.98      0.98      4900
```

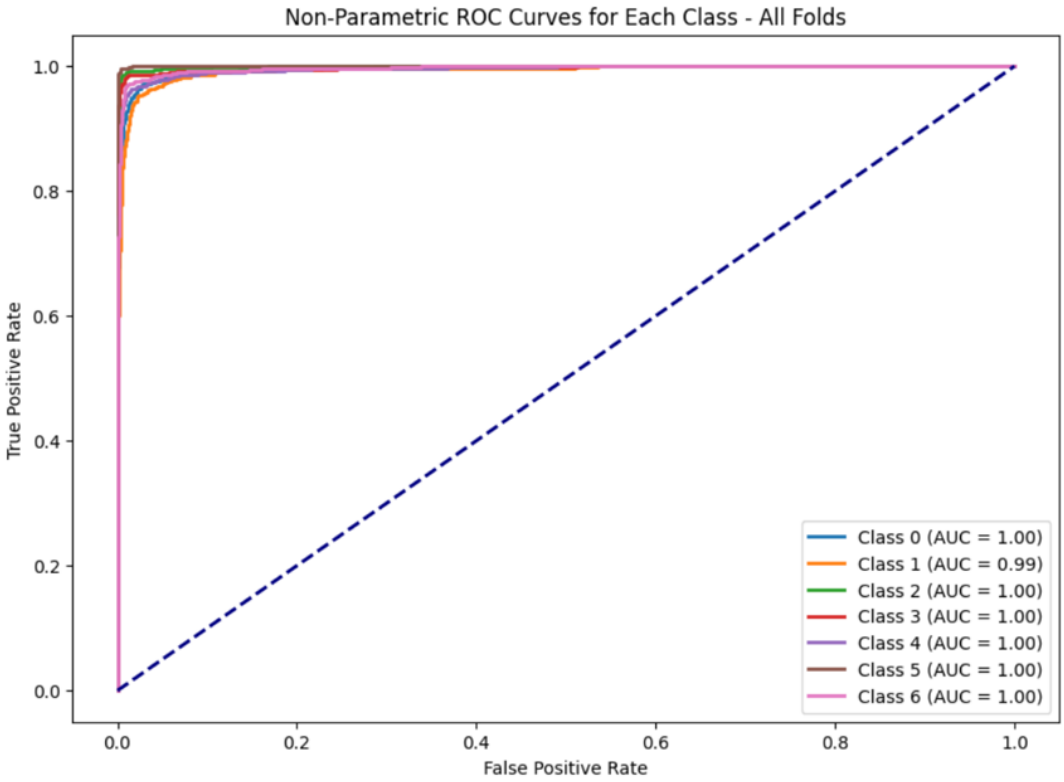


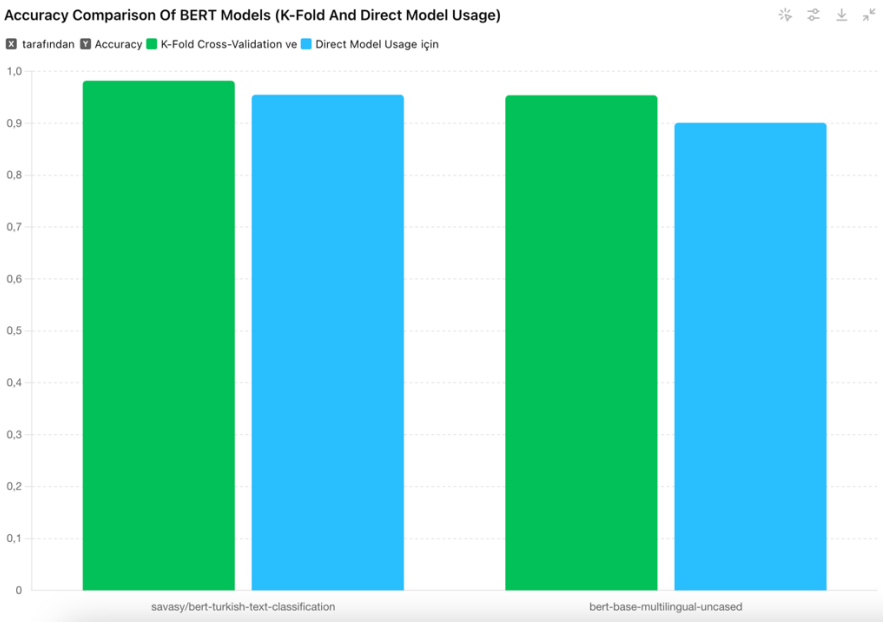


K-fold bert-base-multilingual-uncased:

```
Mean Accuracy: 0.9540816326530613
Mean Precision: 0.9548960926967777
Mean Recall: 0.9540816326530613
Mean F1-Score: 0.9540062535675208

Final Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       700
           1       0.93      0.90      0.91       700
           2       0.97      0.99      0.98       700
           3       0.98      0.96      0.97       700
           4       0.95      0.95      0.95       700
           5       0.98      0.99      0.99       700
           6       0.95      0.96      0.95       700

    accuracy                           0.95      4900
   macro avg       0.95      0.95      0.95      4900
weighted avg       0.95      0.95      0.95      4900
```







6. Conclusion and Evaluation

Best Models:

1. **savasy/bert-turkish-text-classification (K-Fold Cross-Validation)**
   - Accuracy: 0.98204
2. **ANN-BoW Model**
   - Accuracy: 0.9108
3. **SVM - TF-IDF**
   - Accuracy: 0.8943
4. **CNN + BiLSTM (Random Initialization)**
   - **Accuracy** : 0.8711
5. **CNN + BiLSTM (Word2Vec)**
   - **Accuracy :** 0.8469