A Deep Learning Approach for Automated Classification of Refactoring Commits in iOS Applications
iOS Uygulamalarında Refaktörleme Commitlerinin Otomatik Sınıflandırılması için Derin Öğrenme

Hasan Ali Şişeci
Bilgisayar Mühendisliği Bölümü
Cumhuriyet Üniversitesi
Sivas, Türkiye
20239257007@cumhuriyet.edu.tr

***Abstract*— This paper presents an approach for detecting refactor operations in software projects and improving software quality. A total of 5000 commit messages from five different iOS projects were manually labeled as "refactor" and "non-refactor". The labeled data was trained using popular machine learning algorithms such as XGBoost and evaluated on a project-by-project basis. The models used to classify the commit messages were measured with performance metrics such as accuracy, precision, recall and f1-score, and detailed analyses were performed for each project. Challenges encountered during the study, such as data imbalance, and proposed solutions to overcome these challenges are discussed. As a result, it was observed that the methods used for the detection of refactor operations were effective, but performance was limited in some projects due to data imbalance. In the future, this performance can be improved with more balanced datasets and improved models.**

Keywords—*GitHub commit mesajları, refaktör tespiti, makine öğrenimi, XGBoost, veri sınıflandırma*

***Özet*— Bu çalışma, yazılım projelerinde refaktör işlemlerinin tespit edilmesi ve yazılım kalitesinin artırılmasına yönelik bir yaklaşımı ele almaktadır. Beş farklı iOS projesinden elde edilen toplam 5000 commit mesajı, manuel olarak "refaktör" ve "refaktör olmayan" şeklinde etiketlenmiştir. Etiketlenmiş veriler, XGBoost gibi popüler makine öğrenimi algoritmaları kullanılarak eğitilmiş ve projeler bazında ayrı ayrı değerlendirilmiştir. Commit mesajlarının sınıflandırılmasında kullanılan modeller, accuracy, precision, recall ve f1-score gibi performans metrikleri ile ölçülmüş ve her bir proje için detaylı analizler yapılmıştır. Çalışma sırasında karşılaşılan veri dengesizliği gibi zorluklar ile bu zorlukların giderilmesi için önerilen çözümler tartışılmıştır. Sonuç olarak, refaktör işlemlerinin tespiti için kullanılan yöntemlerin etkili olduğu, ancak bazı projelerde veri dengesizliği nedeniyle performansın sınırlı kaldığı gözlemlenmiştir. Gelecekte, daha dengeli veri kümeleri ve gelişmiş modellerle bu performansın artırılabileceği değerlendirilmektedir.**

Anahtar Kelim—GitHub commit mesajları, refaktör tespiti, makine öğrenimi, XGBoost, veri sınıflandırma

I. GİRİS

Yazılım geliştirme süreçlerinde, kodun sürdürülebilirliği, okunabilirliği ve bakım kolaylığı, yazılım kalitesini belirleyen temel unsurlar arasında yer alır. Zamanla karmaşık hale gelen yazılım projelerinde, teknik borç birikimi ve kod kalitesinin düşmesi gibi sorunlar ortaya çıkabilir. Bu tür sorunlarla başa çıkmanın en etkili yollarından biri, mevcut kodun işlevselliğini değiştirmeden, yapısal iyileştirmeler yapmak anlamına gelen refaktör işlemleridir. Refaktör işlemleri, yalnızca yazılım projelerinin yaşam döngüsünü uzatmakla kalmaz, aynı zamanda ekipler arasında iş birliğini kolaylaştırır ve kodun uzun vadede sürdürülebilir olmasını sağlar.

Refaktör işlemleri, genellikle geliştiriciler tarafından commit mesajları aracılığıyla belgelense de, bu bilgilerin sistematik bir şekilde analiz edilmesi manuel yöntemlerle oldukça zordur. Commit mesajlarının birçoğu doğrudan refaktör işlemlerini belirtmediği için, bu mesajların otomatik olarak analiz edilmesi ve sınıflandırılması yazılım mühendisliği açısından kritik bir gereksinim haline gelmiştir. Bu çalışmada, commit mesajları kullanılarak refaktör işlemlerinin tespit edilmesi ve bu sürecin otomatikleştirilmesi hedeflenmiştir.

Bu çalışmanın temel amacı, beş farklı iOS projesinden elde edilen toplam 5000 commit mesajını kullanarak, refaktör işlemlerinin makine öğrenimi algoritmalarıyla etkili bir şekilde tespit edilebileceğini göstermektir. Veriler, manuel olarak "refaktör" ve "refaktör olmayan" olarak etiketlenmiş ve bu etiketler, XGBoost gibi popüler bir sınıflandırma modeli kullanılarak analiz edilmiştir. Model performansı, accuracy, precision, recall ve f1-score gibi metriklerle değerlendirilmiş, projeler arasındaki performans farklılıkları detaylı bir şekilde incelenmiştir.

Araştırma sırasında, özellikle veri dengesizliği gibi yazılım projelerinde sıkça karşılaşılan zorluklar ele alınmış ve bu sorunların çözümüne yönelik çeşitli stratejiler önerilmiştir. Ayrıca, elde edilen sonuçlar, yazılım projelerinde refaktör işlemlerinin tespiti için otomatikleştirilmiş sistemlerin önemini vurgulamaktadır. Çalışma, yazılım mühendisliği topluluğuna kod kalitesini artırmaya yönelik bir rehber sunmayı ve gelecekte yapılacak çalışmalara zemin hazırlamayı amaçlamaktadır.

II. Materyal ve Metod

A. Veri Kümesi

Bu çalışmada kullanılan veri kümesi, GitHub platformundaki açık kaynak iOS projelerinden elde edilmiştir. Araştırmada, projelerin seçiminde belirli kriterler göz önünde bulundurulmuştur. Bu kriterler arasında, projenin popülerliği ve aktifliği açısından önemli olan yıldız (star) sayısı, çatallanma (fork) sayısı ve projeye katkıda bulunan geliştiricilerin (contributors) sayısı yer almıştır. Çalışmaya dahil edilen projelerin her birinin en az 500 yıldız ve 50 katkıcıya sahip olmasına dikkat edilmiştir. Bu tür kısıtlamalar, veri kümesinin çeşitliliğini artırmak ve yazılım geliştirme süreçlerinde refaktör işlemlerinin daha anlamlı bir şekilde incelenmesini sağlamak amacıyla belirlenmiştir.

Veri toplama işlemi, GitHub'ın REST API'si kullanılarak gerçekleştirilmiştir. API aracılığıyla, projelerde yapılan commit mesajları ve ilgili metadata (commit tarihi, yazar bilgisi gibi) toplanmıştır. Çalışmaya beş farklı iOS projesi dahil edilmiş ve her bir projeden 1000 commit mesajı çekilmiştir. Bu mesajlar, Python programlama dili kullanılarak işlenmiş ve anlaşılabilir bir formatta saklanması için CSV dosyalarına dönüştürülmüştür. Böylece toplamda 5000 commit mesajından oluşan kapsamlı bir veri kümesi elde edilmiştir.

Elde edilen veri kümesi, refaktör işlemlerinin tespiti için manuel olarak etiketlenmiştir. Commit mesajları, "refaktör" ve "refaktör olmayan" şeklinde iki ana kategoriye ayrılmıştır. Bu etiketleme işlemi sırasında, commit mesajlarında "refactor", "optimize", "cleanup" gibi terimlere özellikle dikkat edilmiştir. Bu süreç, verinin güvenilir bir şekilde etiketlenmesini sağlamış ve makine öğrenimi modellerinin eğitimi için uygun bir temel oluşturmuştur.

Araştırma sırasında uygulanan bu veri toplama ve işleme adımları, yazılım projelerinde refaktör işlemlerinin sistematik bir şekilde analiz edilmesine yönelik gelecekte yapılacak çalışmalara da yol gösterici niteliktedir. Seçilen projelerin özellikleri ve veri kümesinin kapsamlı yapısı, elde edilen sonuçların genelleştirilebilirliğini artırmak amacıyla özenle tasarlanmıştır.

B. Yöntem

Bu çalışmada, GitHub projelerinden elde edilen commit mesajlarının sınıflandırılması ve refaktör işlemlerinin tespiti amacıyla sistematik bir yöntem geliştirilmiştir. İlk olarak, commit mesajları üzerinde veri ön işleme adımları gerçekleştirilmiştir. Bu adımlar arasında, mesajların temizlenmesi, gereksiz karakterlerin ve durak kelimelerin kaldırılması, küçük harfe dönüştürülmesi ve TF-IDF yöntemi ile vektörleştirilmesi yer almıştır.

Çalışmada, refaktör işlemlerinin doğru bir şekilde tespit edilmesi için XGBoost algoritması tercih edilmiştir. Modelin eğitimi sırasında, commit mesajları manuel olarak "refaktör" ve "refaktör olmayan" şeklinde etiketlenmiş ve bu etiketler modelin öğrenme sürecinde kullanılmıştır. Veri kümesinin %70'i eğitim, %30'u ise test amacıyla ayrılmıştır. Modelin performansı, accuracy, precision, recall ve f1-score gibi metriklerle değerlendirilmiştir.

Son olarak, farklı projelerden elde edilen commit mesajları ayrı ayrı modellenerek projeler arası performans karşılaştırmaları yapılmıştır. Bu karşılaştırmalar, yöntemin genelleştirilebilirliğini ve projelere özgü refaktör örüntülerinin anlaşılmasını sağlamıştır.

III. Uygulama

A. Veri Toplama

Çalışmada kullanılan commit mesajları, GitHub REST API aracılığıyla beş farklı iOS projesinden toplanmıştır. Projelerin seçiminde yıldız (star) sayısı, çatallanma (fork) sayısı ve katkıcı (contributor) sayısı gibi kriterler dikkate alınmıştır. Her bir proje, en az 500 yıldız ve 50 katkıcıya sahip olma şartını sağlamıştır. Ayrıca, projelerin aktif olarak geliştiriliyor olması ve commit geçmişlerinin zenginliği de seçim sürecinde önemli bir rol oynamıştır. API çağrıları ile her projeden 1000 adet commit mesajı ve bunlara ait metadata (commit tarihi, yazar bilgisi vb.) çekilmiştir. Bu veriler, Python kullanılarak CSV formatına dönüştürülmüş ve toplamda 5000 commit mesajından oluşan bir veri kümesi oluşturulmuştur.

B. Veri Ön İşleme

Toplanan commit mesajları, makine öğrenimi modellerine uygun hale getirilmek üzere çeşitli ön işleme adımlarından geçirilmiştir. İlk olarak, mesajlardan özel karakterler, durak kelimeler ve gereksiz boşluklar temizlenmiştir. Daha sonra, metinler küçük harfe dönüştürülerek standart hale getirilmiştir. Kelimelerin önem sıralarını belirlemek ve her bir commit mesajını sayısal bir temsil haline getirmek için TF-IDF (Term Frequency-Inverse Document Frequency) yöntemi kullanılmıştır. Bu yöntem, commit mesajlarının metin tabanlı özelliklerini vektörler şeklinde ifade ederek, makine öğrenimi modelleri için giriş verisi oluşturmuştur.

C. Model Eğitimi

Commit mesajlarının sınıflandırılması için XGBoost algoritması kullanılmıştır. XGBoost, yüksek doğruluk oranı ve hız avantajları nedeniyle tercih edilmiştir. Veri kümesi, %70 eğitim ve %30 test verisi olarak ikiye ayrılmıştır. Eğitim sırasında, model parametreleri (örneğin, öğrenme oranı, maksimum derinlik) hyperparameter tuning ile optimize edilmiştir. Eğitim verisi, manuel olarak "refaktör" ve "refaktör olmayan" şeklinde etiketlenmiş commit mesajlarını içermektedir. Model, bu etiketleri kullanarak sınıflandırmayı öğrenmiş ve test verisi üzerindeki performansı ölçülmüştür.

D. Model Değerlendirme ve Karşılaştırma

Eğitim süreci tamamlandıktan sonra, modelin performansı test verileri üzerinde değerlendirilmiştir. Performans ölçütü olarak accuracy, precision, recall ve f1-score metrikleri kullanılmıştır. Ayrıca, confusion matrix ile modelin doğru ve yanlış tahminleri görselleştirilmiştir. Çalışmada her bir proje ayrı ayrı modellenmiş ve projeler arası performans karşılaştırmaları yapılmıştır. Bu analizler, yöntemin genelleştirilebilirliğini ve farklı projelerdeki refaktör örüntülerini anlamak için önemli bir veri sağlamıştır.

IV. SONUÇLAR

Bu çalışmada, yazılım projelerinde refaktör işlemlerinin tespiti amacıyla, GitHub'dan elde edilen commit mesajlarının makine öğrenimi temelli bir yöntemle analiz edilmesi hedeflenmiştir. Beş farklı iOS projesinden elde edilen commit mesajları, manuel olarak etiketlenmiş ve XGBoost algoritması ile sınıflandırılmıştır. Çalışmanın sonucunda, yöntem hem genel hem de proje bazında değerlendirilmeye tabi tutulmuştur.

A. Genel Bulgular

Modelin genel performansı, accuracy, precision, recall ve f1-score gibi metriklerle analiz edilmiştir. Genel olarak, model "Non-Refactor" commitlerini yüksek doğrulukla sınıflandırmış; ancak, veri dengesizliği nedeniyle "Refactor" commitlerinde performans değişiklikleri gözlemlenmiştir. Özellikle, bazı projelerde "Refactor" commitlerinin düşük sayıda olması, modelin bu sınıfı öğrenme yeteneğini sınırlamış ve recall değerlerini düşürmüştür. Bu durum, veri kümesinin dengesizliğinden kaynaklanan bir problem olarak öne çıkmıştır.

B. Proje Bazlı Değerlendirme

Signal Projesi: Bu projede modelin genel doğruluğu yüksek olmasına rağmen "Refactor" commitlerini doğru sınıflandırma oranı oldukça düşüktür. Precision ve recall değerlerinin sıfır olması, modelin tamamen "Non-Refactor" commitlerine odaklandığını göstermektedir. Bu durum, veri dengesizliği problemini açık bir şekilde ortaya koymaktadır.

Firefox Projesi: Modelin performansı oldukça dengeli ve yüksek bulunmuştur. Recall değeri %91,8 ve f1-score %95,7 olarak hesaplanmıştır. Bu proje, modelin "Refactor" commitlerini yüksek doğrulukla tespit edebildiği bir örnek olarak öne çıkmıştır.

Wikipedia Projesi: Accuracy %98,6 ile en yüksek değeri göstermiştir. Ancak, recall değerinin %87,5 olması, modelin "Refactor" commitlerinde bazı örnekleri kaçırdığını göstermektedir. Bununla birlikte, precision değerinin %100 olması, modelin tahminlerinin güvenilirliğini artırmaktadır.

Lottie Projesi: Bu projede model, "Refactor" commitlerini tespit etmede sınırlı bir başarı göstermiştir. Recall %45,5 ile düşük seviyede kalmış; ancak, precision %90,9 ile tahminlerin büyük ölçüde doğru olduğunu göstermiştir. Veri dengesizliği, model performansını olumsuz etkilemiştir.

Brave Projesi: Bu projede model, genel olarak dengeli bir performans sergilemiştir. Recall %82,9, precision %94,4 ve f1-score %88,3 olarak hesaplanmıştır. "Refactor" commitlerinin sınırlı sayıda olmasına rağmen model, bu commitleri doğru bir şekilde tespit edebilmiştir.

C. Tartışma

Sonuçlar, yöntemin genel olarak "Non-Refactor" commitlerini yüksek doğrulukla tespit edebildiğini; ancak, "Refactor" commitlerinin sınırlı sayıda olduğu durumlarda performansın düştüğünü göstermektedir. Özellikle Signal ve Lottie projelerinde görülen düşük recall değerleri, veri dengesizliğinin model performansı üzerindeki etkisini açıkça ortaya koymaktadır. Bu durum, veri kümesinin dengesini sağlamak için veri artırma yöntemlerinin kullanılmasının önemini vurgulamaktadır.

Firefox ve Wikipedia projelerinde gözlemlenen yüksek performans, yöntemin etkili bir şekilde uygulanabileceğini göstermektedir. Bununla birlikte, Brave projesi, az sayıda "Refactor" commitine rağmen modelin bu commitleri öğrenebildiğini ortaya koymaktadır. Bu durum, yöntemin genelleştirilebilirlik potansiyelini desteklemektedir.

V. GELECEK ÇALIŞMALAR

Bu çalışma, iOS projelerinde commit mesajlarını analiz ederek refaktör işlemlerinin tespit edilmesine yönelik bir yöntem sunmuştur. Beş farklı iOS projesi üzerinden yapılan analizler, yöntemin genel olarak etkili olduğunu ancak veri dengesizliği gibi faktörlerin model performansını olumsuz etkileyebileceğini göstermiştir. Bu sonuçlar, yazılım mühendisliği alanında refaktör tespiti için otomatikleştirilmiş yaklaşımların önemini vurgulamaktadır.

Gelecekte yapılacak çalışmalar, bu akademik araştırmanın sunduğu yöntemi daha geniş bir bağlamda ele almayı amaçlayabilir. Bu bağlamda, aşağıdaki yönlendirmeler ve geliştirmeler önerilmektedir:

A. Android Projelerinin Analizi:

Aynı yöntem, Android projelerinde de uygulanarak farklı platformlardaki commit mesajlarının benzerlikleri ve farklılıkları analiz edilebilir. Bu analiz, iOS ve Android projelerindeki refaktör işlemlerinin örüntülerini ve bu işlemlerin yazılım geliştirme süreçlerine etkisini anlamada faydalı olacaktır. Platformlar arası karşılaştırmalar, refaktör işlemlerinin yazılım mühendisliği pratikleri üzerindeki genel etkilerini incelemek için önemli bir fırsat sunar. Örneğin:

iOS projelerinde kullanılan refaktör işlemlerinin, Android projelerinde nasıl farklılık gösterdiği,
Commit mesajlarında kullanılan dilin platforma özgü yapılar içerip içermediği,
Veri dengesizliğinin her iki platformda da benzer şekilde model performansını etkileyip etkilemediği incelenebilir.
Veri Dengesinin İyileştirilmesi:
Signal ve Lottie projelerinde gözlemlendiği gibi, "Refactor" commitlerinin sınırlı sayıda olduğu veri kümelerinde, veri artırma yöntemlerinin (örneğin, SMOTE) etkisi araştırılabilir. Özellikle, Android projelerinde de benzer bir dengesizlik gözlemlenirse, bu tür yaklaşımlar her iki platformda karşılaştırmalı olarak test edilebilir.

B. Commit İçeriklerinin Dahil Edilmesi:

Commit mesajlarının yanı sıra, commit içeriği (örneğin, kod değişiklikleri, dosya isimleri) gibi ek özelliklerin modele dahil edilmesi ile sınıflandırma doğruluğu artırılabilir. Bu tür bir genişleme, hem iOS hem de Android projelerinde refaktör tespitine yönelik daha kapsamlı bir analiz sağlar.

C. Gelişmiş Modellerin Test Edilmesi:

XGBoost gibi güçlü makine öğrenimi yöntemleri bu çalışmada etkili bir şekilde kullanılmış olsa da, derin öğrenme tabanlı modeller (örneğin, LSTM, BERT) ile commit mesajlarının dil yapısını daha derinlemesine anlamak mümkün olabilir. Hem iOS hem de Android projelerinde bu tür modeller test edilerek performans karşılaştırması yapılabilir.

D. Platformlar Arası Genel Çıkarımlar:

iOS ve Android projelerinde elde edilen sonuçlar birleştirilerek platformlar arasındaki refaktör dinamikleri karşılaştırılabilir. Örneğin:

Hangi platformda daha fazla refaktör işlemi yapılmaktadır? Platformların commit mesajlarında kullanılan dil ve terminoloji açısından farkları nelerdir? Refaktör işlemlerinin projelere etkisi, platformlar arasında nasıl farklılık göstermektedir?

Bu tür karşılaştırmalar, yazılım mühendisliği pratiklerinin platformlar arasında nasıl farklılık gösterdiğini anlamada önemli bilgiler sunabilir. Ayrıca, elde edilen sonuçlar, her iki platform için daha genelleştirilebilir refaktör tespit yöntemlerinin geliştirilmesine katkıda bulunabilir.

Sonuç olarak, bu çalışma, iOS projelerinde etkili bir refaktör tespit yöntemi sunmakla kalmamış, aynı zamanda yazılım geliştirme süreçlerinde platformlar arası analizlere yönelik bir zemin oluşturmuştur. Gelecekte yapılacak çalışmalar, önerilen yöntemlerin daha geniş veri kümelerinde ve farklı platformlarda uygulanmasını sağlayarak yazılım mühendisliği topluluğuna daha kapsamlı katkılar sunabilir.

VI. KAYNAKLAR

[1] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794, 2016. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[2] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” Information Processing & Management, vol. 24, no. 5, pp. 513–523, 1988. [Online]. Available: https://doi.org/10.1016/0306-4573(88)90021-0

[3] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002. [Online]. Available: https://doi.org/10.1613/jair.953

[4] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543, 2014. [Online]. Available: https://doi.org/10.3115/v1/D14-1162

[5] GitHub Inc., “GitHub REST API Documentation,” [Online]. Available: https://docs.github.com/rest

[6] C. Bishop, Pattern Recognition and Machine Learning, 1st ed. Springer, 2006. [Online]. Available: https://www.springer.com/gp/book/9780387310732

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in Proceedings of the International Conference on Learning Representations (ICLR), 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[8] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4171–4186, 2019. [Online]. Available: https://doi.org/10.18653/v1/N19-1423

[10] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, “A neural probabilistic language model,” Journal of Machine Learning Research, vol. 3, pp. 1137–1155, 2003. [Online]. Available: https://dl.acm.org/doi/10.5555/944919.944966