Applying the AI Development Workflow

AI for Software Engineering

Name:- Abemelek Samson Abduke

Submission Date: Nov 6, 2025

Part 1: Short Answer Questions

1. Problem Definition

- **Hypothetical AI Problem:** Predicting customer churn for a monthly subscription streaming service.
- **Objectives (3):**
    1. Proactively identify subscribers who are at high risk of canceling their subscription in the next 30 days.
    2. Enable the marketing team to target at-risk users with tailored retention offers (e.g., discounts, content recommendations).
    3. Reduce the overall monthly churn rate and increase customer lifetime value.
- **Stakeholders (2):**
    1. **Marketing Team:** Needs the list of at-risk users to design and execute retention campaigns.
    2. **Product Team:** Needs to understand *why* users are churning (e.g., low engagement, high buffering rates) to inform product improvements.
- **Key Performance Indicator (KPI) (1):**
    1. **Monthly Churn Rate:** The percentage of subscribers who cancel their service in a given month. The model's success will be measured by its ability to help decrease this KPI.

2. Data Collection & Preprocessing

- **Data Sources (2):**
    1. **User Activity Database:** Contains granular data on user behavior, such as *login_frequency, hours_watched_per_week, content_genres_viewed, and devices_used.*
    2. **Customer Account Database:** Contains static data about the user, such as *subscription_plan_type, account_tenure* (how long they've been a customer), *billing_history, and demographic_information* (if collected).
- **Potential Bias (1):**
    1. **Historical Bias:** If, in the past, the service had significant technical issues (e.g., high buffering) that primarily affected users in a specific region, the data will show a high correlation between that region and churn. A model trained on this data might learn to flag *all* users from that region as "high-risk," even if the technical issues have been solved. This unfairly penalizes users based on location.
- **Preprocessing Steps (3):**
    1. **Handling Missing Data:** Many users might have missing *hours_watched* data (if they signed up but never watched). Instead of dropping them (which is a strong signal of churn), we would impute this with 0.
    2. **Feature Encoding:** Convert categorical features into numbers. For example, *subscription_plan_type* ('Basic', 'Premium', 'Family') would be one-hot encoded into separate binary columns.
    3. **Feature Scaling:** Numerical features like *account_tenure* (e.g., 1-60 months) and *hours_watched_per_week* (e.g., 0-40 hours) are on different scales. We would use **StandardScaler** to normalize them, ensuring one feature doesn't disproportionately influence the model.

3. Model Development

- **Model Choice & Justification:**
    1. **Model: Logistic Regression**.
    2. **Justification:** For a churn problem, *why* a customer is leaving is as important as *if* they are leaving. Logistic Regression is a simple, fast, and highly **interpretable** model. After training, we can easily inspect the model's coefficients to see exactly which features (e.g., *login_frequency* or *account_tenure*) are the strongest predictors of churn. This provides actionable insights to the Product and Marketing teams.
- **Data Splitting:**
    1. The data would be split into three sets:
        1. **Training Set (70%):** The largest portion, used to teach the model to find patterns between user behavior and churning.
        2. **Validation Set (15%):** Used to tune hyperparameters. The model is evaluated against this set repeatedly to find the best settings without "cheating" by seeing the test data.
        3. **Test Set (15%):** Used *only once* at the very end to get a final, unbiased assessment of the model's real-world performance.
- **Hyperparameters to Tune (2):**
    1. ***C*** **(Regularization Strength):** This controls the penalty for model complexity. Tuning this (e.g., 0.1, 1, 10) is crucial to prevent **overfitting**. A smaller *C* value creates a simpler model that may generalize better.
    2. ***solver*****:** This is the algorithm used to find the optimal model parameters. Tuning this (e.g., 'liblinear', 'saga') can significantly speed up training time and find a better solution, especially on large datasets.

4. Evaluation & Deployment

- **Evaluation Metrics (2):**
    - **Recall (Sensitivity):** Calculated as TP / (TP + FN). This measures: "Of all the customers who *actually* churned, what percentage did our model correctly identify?" **This is our most important metric** because a False Negative (missing a churner) is very costly—we lose that customer forever.
    - **Precision:** Calculated as TP / (TP + FP). This measures: "Of all the customers our model *predicted* would churn, what percentage actually did?" This is important for the Marketing team so they don't waste retention offers (which cost money) on happy customers.
- **Concept Drift & Monitoring:**
    - **Concept Drift** is the phenomenon where the statistical properties of the target variable (churn) change over time. The patterns that predicted churn 6 months ago might no longer be relevant (e.g., a price hike, a new competitor, or a hit new show could all change *why* people leave).
    - **Monitoring:** We would monitor this by running the model in "shadow mode," comparing its predictions on new, live data against the actual outcomes. We would set up alerts to trigger if the model's **Recall** drops below a pre-defined threshold (e.g., 85%). This signals that the model is drifting and needs to be retrained on more recent data.
- **Technical Deployment Challenge (1):**

    - **Scalability & Latency:** The streaming service may have millions of users. The deployment infrastructure must be able to run predictions for all users in a timely manner (e.g., a nightly batch job). If the goal is *real-time* prediction (e.g., "show a retention offer *as* they try to click 'cancel'"), the challenge becomes building a low-latency API (e.g., using FastAPI) that can return a prediction in milliseconds, which requires significant engineering effort.

Part 2: Case Study Application (Hospital Readmission)

1. Problem Scope

- **Problem:** A hospital is facing high rates of patient readmission within 30 days of discharge, particularly for patients with chronic conditions. This leads to poor patient outcomes, strained hospital resources, and potential financial penalties from insurers.
- **Objectives:**
    1. Accurately identify patients who are at high risk of readmission *before* they are discharged.
    2. Allow hospital staff (e.g., case managers) to deploy targeted post-discharge interventions (e.g., follow-up calls, home nurse visits) to these high-risk patients.
    3. Reduce the 30-day readmission rate by 15% within the first year.
- **Stakeholders:**
    1. **Clinical Staff (Doctors, Nurses):** Need a reliable tool to help them prioritize patient follow-up care.
    2. **Hospital Administrators:** Need to reduce costs associated with readmissions and comply with healthcare regulations.
    3. **Patients and Families:** Want to avoid the stress and health risk of a return to the hospital.
    4. **IT Department:** Must integrate and maintain the tool within existing hospital systems.

2. Data Strategy

- **Data Sources:**
    1. **Electronic Health Records (EHRs):** Patient's primary diagnosis, co-morbidities (e.g., diabetes, heart failure), lab results (e.g., A1c, creatinine levels), vital signs.
    2. **Admission/Discharge/Transfer (ADT) Data:** Length of initial hospital stay, admission type (e.g., 'Emergency', 'Elective'), number of previous admissions.
    3. **Patient Demographics:** *age, gender,* and (with extreme caution) *zip code* (as a proxy for socioeconomic factors).
    4. **Medication Records:** List of prescribed medications at discharge.
- **Ethical Concerns (2):**
    1. **Patient Privacy (HIPAA):** The model would be trained on Protected Health Information (PHI). All data must be de-identified before being accessed by the data science team, and the final model must be deployed on a secure, HIPAA-compliant server with strict access controls and audit logs.
    2. **Algorithmic Bias:** The model could inadvertently learn and perpetuate existing health inequities. For example, if patients from a certain

socioeconomic background (proxied by *zip code*) historically had higher readmission rates due to a *lack of access* to follow-up care, the model might flag this demographic as inherently "high-risk." This could lead to a feedback loop where this group is unfairly labeled, rather than the *root cause* (lack of resources) being addressed.

- **Preprocessing Pipeline & Feature Engineering:**
    1. **Data Integration:** Join the various data sources (EHR, ADT, Demographics) on a unique, de-identified *patient_ID*.
    2. **Feature Engineering:** This is critical. We would create new, informative features:
        - *length_of_stay*: (Date of discharge - Date of admission).
        - *comorbidity_score*: A count of the number of chronic conditions the patient has (e.g., from ICD-10 diagnosis codes).
        - *num_prior_admissions*: A count of hospital admissions in the last 12 months.
        - *medication_count*: The total number of unique medications prescribed at discharge.
    3. **Handling Missing Data:** Use domain-specific imputation. For example, a missing lab value might be imputed with the mean *for that patient's age group*, not the overall mean.
    4. **Encoding & Scaling:** One-hot encodes categorical features like *admission_type* ('Emergency', 'Elective', 'Urgent'). Standardize numerical features like *age, length_of_stay, and comorbidity_score.*

3. Model Development

- **Model Choice & Justification:**
    - **Model: Random Forest**.
    - **Justification:** This is a strong, balanced choice for this problem.
        - **Accuracy:** It is a powerful ensemble model that can capture complex, non-linear relationships in medical data (e.g., the risk from *age* might only be high *if comorbidity_score* is also high).
        - **Interpretability:** While not as simple as Logistic Regression, a Random Forest provides **Feature Importance**. This allows us to show doctors *which* factors were most influential in the model's decision (e.g., "Risk is high due to: *num_prior_admissions and lab_result_X"*). This builds trust and makes the prediction actionable.
        - **Robustness:** It handles a mix of numerical and categorical data well and is less sensitive to outliers than some other models.
- **Hypothetical Confusion Matrix & Metrics:**
    - *Scenario:* We test our model on a test set of 200 patients. Of these, 40 were actually readmitted (Positive) and 160 were not (Negative).
    - **Hypothetical Results:**
        - **True Positives (TP):** 32 (We correctly predicted 32 of the 40 readmissions)
        - **False Negatives (FN):** 8 (We missed 8 readmissions; this is the worst outcome)

        - **False Positives (FP):** 16 (We predicted 16 patients would be readmitted, but they were fine)
        - **True Negatives (TN):** 144 (We correctly predicted 144 patients would not be readmitted)
    - Confusion Matrix:

	Predicted: Readmit	Predicted: No Readmit
Actual: Readmit	32 (TP)	8 (FN)
Actual: No Readmit	16 (FP)	144 (TN)

    - **Calculations:**
        - **Precision** = TP / (TP + FP) = 32 / (32 + 16) = 32 / 48 = **66.7%**
            - *Interpretation:* When our model flags a patient as high-risk, it is correct 66.7% of the time.
        - **Recall** = TP / (TP + FN) = 32 / (32 + 8) = 32 / 40 = **80%**
            - *Interpretation:* Our model successfully identified 80% of all patients who were *actually* readmitted. This is a good starting point, as catching the most at-risk patients (high Recall) is the primary goal.

4. Deployment

- **Integration Steps:**
    1. **Containerize the Model:** Package the trained model, preprocessing pipeline, and all dependencies into a Docker container.
    2. **Create a Secure API:** Build a lightweight, secure REST API (e.g., using Flask or FastAPI) that exposes a single endpoint (e.g., */predict*). This endpoint accepts a patient's data (in JSON format) and returns a risk score (e.g., *{"risk_score": 0.85, "top_factors": ["prior_admissions", "lab_result_x"]}).*
    3. **Deploy to Secure Server:** Deploy this container on the hospital's internal, HIPAA-compliant cloud or on-premise server.
    4. **EHR Integration:** The hospital's IT team integrates this API into the existing EHR software. When a doctor opens the discharge workflow for a patient, the EHR automatically sends that patient's data (anonymously) to the API and displays the returned risk score (e.g., "85% High Risk") and the top reasons, seamlessly within the doctor's existing view.
- **HIPAA Compliance:**
    1. **Data Minimization:** The API is designed to *only* receive the minimum data necessary for a prediction. It does not receive or store the patient's name, SSN, or other direct identifiers.
    2. **Encryption:** All data is encrypted in transit using HTTPS/SSL and at rest on the server.

3. **Access Control:** The API endpoint is not public. It is firewalled and only accessible from authorized IP addresses within the hospital's internal network (i.e., the EHR system).
4. **Audit Logs:** Every single request to the API is logged for auditing purposes to track who/what accessed patient data and when.

5. Optimization

- **Method to Address Overfitting:**
    1. **Hyperparameter Tuning (Pruning):** For our Random Forest, overfitting occurs when the individual trees become too complex and "memorize" the training data. The best way to combat this is to **tune pruning parameters**. We would use a technique like **Grid Search** or **Randomized Search** to find the optimal values for:
        - *max_depth:* Limiting the maximum depth of each tree.
        - min_samples_leaf: Increasing the minimum number of samples required to be at a leaf node.
          This forces the model to learn simpler, more generalizable patterns, which will improve its performance on unseen data.

Part 3: Critical Thinking

1. Ethics & Bias

- **How Biased Data Affects Outcomes:** Biased training data in this case study would have catastrophic patient outcomes. If the model learns from historical data that "patients who speak English as a second language" (a feature that might be in the EHR) are readmitted more often (perhaps due to poor-quality translated discharge instructions), it will flag *all* such patients as high-risk. A busy clinical team, trusting the AI, might (even subconsciously) treat these patients differently or allocate resources based on this biased flag. This creates a *discriminatory feedback loop* where a specific group receives a different standard of care not because of their medical condition, but because of their language, directly worsening health inequality.
- **Strategy to Mitigate Bias:**
    1. **Bias Auditing and Fairness-Aware Modeling:**
        - Before deployment, we must **audit the model for bias**. This involves testing its performance (e.g., its False Negative Rate) *across* different protected groups (e.g., race, gender, language).
        - If the model is found to be biased (e.g., it is much less accurate for women than for men), we must intervene.
        - One strategy is to use **Fairness-Aware Modeling techniques** (e.g., "Adversarial Debiasing" or applying "Demographic Parity" constraints). This involves re-training the model with a secondary objective: not only to be *accurate*, but also to be *equally accurate* across all groups. This forces the model to learn medical patterns, not discriminatory social proxies.

2. Trade-offs

- **Interpretability vs. Accuracy:**

    - In healthcare, there is a strong preference for **interpretability** (or "explainability") over a small increase in **accuracy**.
    - A highly complex "black box" model (like a deep neural network) might be 90% accurate. A simpler, interpretable model (like Logistic Regression or a Random Forest with feature importance) might be 88% accurate.
    - Most hospitals would choose the 88% accurate model. **Why?** A doctor must be able to trust and act on the AI's recommendation. If the model simply says "High Risk" with 90% confidence, the doctor has no way to verify it or know *what* to do. If the 88% model says "High Risk, because of *creatinine_level and num_prior_admissions*," the doctor can immediately validate this against the patient's chart and tailor an intervention (e.g., "This patient needs a post-discharge kidney function check and a home nurse visit."). An unexplainable model is unactionable and, therefore, useless (or even dangerous) in a clinical setting.
- **Limited Computational Resources:**
    - If the hospital has limited computational resources (e.g., no GPUs, only standard CPU servers), this **immediately rules out deep learning models** (Neural Networks), which are computationally expensive to train.
    - This constraint *forces* the model choice towards more "classic" machine learning algorithms that are highly efficient to train on CPUs. This would strongly favor models like:
        1. **Logistic Regression** (very fast to train)
        2. **Random Forest** (can be parallelized on multiple CPU cores)
        3. **Gradient Boosting (e.g., LightGBM)** (often provides the best performance on tabular data and is highly optimized for CPU usage).
    - Therefore, a lack of resources pushes the team away from "bleeding-edge" models and towards practical, efficient, and often more interpretable solutions.

Part 4: Reflection & Workflow Diagram

1. Reflection

- **Most Challenging Part of the Workflow:**
    - The most challenging part of the AI development workflow, especially in a real-world context, is **Data Collection and Preprocessing**. In the case study, the required data is not in a single, clean spreadsheet. It is fragmented across multiple, complex, and highly protected systems (EHRs, labs, pharmacy, ADT).
    - The challenge is not just *technical* (joining tables, cleaning messy data) but also *organizational* and *ethical*. It requires getting approval from legal, compliance (HIPAA), and IT departments; navigating data access; and ensuring every step is secure and de-identified. This stage often takes 80% of the total project time and is far more difficult than the "glamorous" work of model training.
- **Improvement with More Time/Resources:**
    - With more time and resources, the single biggest improvement would be to **incorporate unstructured data using Natural Language Processing (NLP)**.

- Currently, our model only uses structured data (lab values, diagnosis codes). However, the most valuable insights are often hidden in the unstructured text of **"doctors' notes"** or **"nurses' discharge comments"**.
- Notes like "patient lives alone," "seems confused about medication," or "lacks family support" are *extremely* high predictors of readmission. With more resources, I would build an NLP pipeline to extract these concepts from the text and add them as features to the model, which would almost certainly improve its accuracy and value significantly.

2. Diagram

Here is a flowchart of the complete, cyclical AI Development Workflow, labeling all key stages.

graph TD

A[Problem Definition] --> B[Data Collection]

B --> C[Data Preprocessing]

C --> D[Model Development & Training]

D --> E[Model Evaluation]

E --> F[Deployment]

F --> G[Monitoring & Maintenance]

G --> A[Feedback Loop for Retraining]

References

1. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
2. Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.
3. U.S. Department of Health & Human Services. (n.d.). *Health Information Privacy*. HHS.gov. Retrieved from https://www.hhs.gov/hipaa/index.html
4. Jurafsky, D., & Martin, J. H. (2023). *Speech and Language Processing* (3rd ed.). Prentice Hall.