ChatGPT

Domain-Specific Machine Learning Project Outlines

1. Healthcare – Federated Learning

Objective: Develop a **federated learning** model for a healthcare application, enabling multiple hospitals or devices to collaboratively train a machine learning model without sharing sensitive patient data. Federated learning allows models to achieve performance comparable to centrally trained models while preserving data privacy [1]. For example, hospitals could jointly train a model to detect a certain disease from medical images or health records, with each hospital's data kept locally. This approach is particularly appealing in healthcare, where privacy regulations (like HIPAA) restrict data sharing, yet pooling knowledge can improve diagnostic accuracy [2] [3].

Key Challenges: - *Data Privacy and Security:* Ensuring no sensitive patient data is exposed. Only model updates (gradients) are shared, often with added privacy techniques like differential privacy or secure aggregation [4]. - *Heterogeneity:* Different hospitals may have **non-IID data** (varying patient demographics or devices) leading to convergence challenges [5]. - *Communication and Efficiency:* Limited network bandwidth and the need to aggregate updates from many clients. The framework must handle dropped clients or asynchronous updates.

Publicly Available Dataset(s): One can simulate federated learning on public medical data. For example, the **MIMIC-III** critical care database (available via PhysioNet) provides ICU patient records and is often used in federated experiments [6]. Researchers also use public medical imaging datasets like NIH's Chest X-ray set (14 diseases) or the BraTS brain tumor MRI dataset, partitioning by institution to mimic siloed data. The **MIT-BIH Arrhythmia** dataset (ECG signals) is another standard – it's been used in federated studies for cardiac signal classification [7]. These datasets allow experimenting with federated training while preserving realism in data distribution.

Methods and Tools: The project would use federated averaging (FedAvg) as a baseline algorithm, where a central server coordinates model updates from distributed clients. **Open-source frameworks** are available to simplify implementation: - *TensorFlow Federated (TFF):* An open-source framework for training models on decentralized data [8]. - *PySyft:* A Python library for federated learning and privacy-preserving deep learning [9]. - *Flower:* A framework-agnostic federated learning library that can integrate with PyTorch or TensorFlow [10].
Using these, one can train models (e.g., a CNN for images or an LSTM for time-series EHR data) across simulated client nodes. Standard deep learning libraries (PyTorch, TensorFlow) will be used for model definition, and federated frameworks handle the distributed training loop. Additional tools like Docker or gRPC can simulate multiple hospital nodes in a lab environment.

Evaluation Metrics: Model performance is measured using the same metrics as centralized learning, depending on the task. For classification (e.g., disease vs. no disease), use **accuracy, precision, recall,** and **F1-score** on a held-out test set. Because data is siloed, evaluating generalization across institutions is key –

e.g., measuring performance on each hospital's data to ensure the federated model generalizes well. If the task is regression (e.g., predicting a health risk score), use **Mean Squared Error (MSE)** or **R²**. Additionally, one may compare federated model metrics to a hypothetical centrally-trained model as a benchmark. It has been shown that federated models can reach similar accuracy and precision as central models [4]. In federated settings, one might also track **communication rounds** until convergence or **communication cost** (bytes sent) as an efficiency metric.

Expected Outcomes: By project's end, students will have a **working federated learning prototype** (e.g., a model that detects a condition from patient data) that achieves near-centralized performance while keeping data distributed. They can expect to produce a **report** detailing how the federated model's accuracy compares to a traditional approach, and how privacy was preserved. The outcome should demonstrate that collaboration between institutions is possible without direct data sharing, for instance: *"The federated model for X-ray pneumonia detection reached ~95% of the accuracy of a pooled-data model, while each hospital's data stayed local."* [4]. This project teaches practical knowledge of privacy-preserving AI, and highlights any trade-offs (like slightly higher training time or communication overhead) for the benefit of patient privacy.

2. Healthcare – Personalized Medicine Model

Objective: Build a machine learning model to assist in **personalized medicine**, tailoring medical decisions or treatments to individual patient characteristics. The goal is to predict the most effective treatment or diagnosis for a specific patient based on their personal data – for example, genomic information, medical history, lab results, etc. This could involve predicting a patient's response to a medication, or classifying subtypes of a disease for targeted therapy. In essence, the model helps move from a "one-size-fits-all" approach to **personalized treatment recommendations**. For instance, in oncology the model might classify genetic mutations in a tumor to decide which drug will be most effective [11]. Personalized models promise improved outcomes by accounting for individual variability in genes or lifestyle.

Key Challenges: - *Data Richness:* Personalized medicine often requires integrating diverse data (genomic sequences, lab tests, clinical notes), leading to high-dimensional feature spaces and smaller sample sizes per feature. - *Interpretability:* Clinicians need to trust recommendations. The model should ideally provide interpretable outputs (e.g., which biomarkers influenced the decision) to support clinical adoption. - *Privacy and Ethics:* Personal health data (especially genetics) is sensitive. Projects must handle data securely and consider ethical implications of predictive modeling in healthcare (e.g., avoiding biases).

Publicly Available Dataset(s): A well-known example is the **Personalized Medicine: Redefining Cancer Treatment** dataset from a Kaggle competition (Memorial Sloan Kettering, 2017). It contains genomic mutations and text evidence from research articles, with mutations labeled into 9 classes based on clinical effect [12]. Using this dataset, one can train a model to classify mutations (e.g. "driver" vs "passenger" mutations) – a step toward personalized cancer therapies. Another dataset is **TCGA (The Cancer Genome Atlas)**, which provides genomic profiles for various cancers along with patient outcomes; students could use a subset to predict treatment response. For pharmacogenomics (drug response prediction), the **GDSC** or **Cell Line** datasets (public databases mapping genetic features to drug sensitivities) are available. For example, the **GDSC** dataset could allow a model to predict how a tumor cell line with certain mutations responds to a panel of drugs, informing personalized drug choice. These public datasets enable building and evaluating models in a controlled setting before clinical data is used.

Methods and Tools: Depending on data type, a mix of machine learning and deep learning techniques will be used: - *If genomic or tabular data:* Tree-based models (Random Forests, XGBoost) or neural networks can identify important features (e.g., specific gene mutations). Feature selection or dimensionality reduction (PCA, autoencoders) might be needed due to high dimensionality of genomic data. - *If text (clinical notes or literature):* NLP models like BERT (or BioBERT, a biomedical variant) can be fine-tuned to classify text (for instance, classifying patient records or research evidence in support of a mutation's effect [11]). - *Tools & Libraries:* **Scikit-learn** is useful for prototyping classification/regression models on tabular clinical data. **TensorFlow** or **PyTorch** will be used for deep learning – e.g., building neural networks that take patient features as input. For genomic data, libraries like **pandas** (for data manipulation) and **scikit-bio** or **BioPython** (for genomic sequence processing) can help. If the project involves NLP on clinical text, **Hugging Face Transformers** can provide pre-trained biomedical language models. Additionally, specialized frameworks exist, such as **TensorFlow Probability** for incorporating probabilistic models (to capture uncertainty, which is valuable in medical predictions).

Evaluation Metrics: The evaluation depends on the prediction task: - For classification (e.g., predicting responder vs non-responder to a treatment, or mutation effect classes), use **accuracy** and **F1-score**, but also **precision/recall**. In medicine, recall (sensitivity) might be crucial – e.g., capturing all patients who would benefit from a drug – while precision ensures we don't mis-identify patients who won't benefit. **ROC-AUC** is useful if predicting risk scores or doing binary classification with imbalanced classes (e.g., rare mutation effects). - For regression (e.g., predicting an optimal drug dosage or a risk score), use **Mean Absolute Error (MAE)** or **Mean Squared Error (MSE)** to measure prediction error against actual outcomes. - Importantly, evaluation should consider **clinical relevance**: for example, does the model identify subgroups of patients correctly? If multiple classes (like mutation functional classes), a **confusion matrix** can show if certain classes are often mistaken for others (which might indicate biological similarity). - If the dataset is imbalanced (common in genomics where, say, only a few mutations are "drivers"), **balanced metrics** like macro-averaged F1 or area under Precision-Recall curve are appropriate. For instance, in the cancer mutation classification, the model's performance was measured by its accuracy in predicting the correct mutation category out of 9 classes [11]. - Sometimes, **domain-specific metrics** appear: e.g., in dosage prediction, one might measure the percentage of patients for whom the dose was within a clinically acceptable range.

Expected Outcomes: Students will deliver a **predictive model and analysis** that demonstrates how personalized predictions can be made. For example, the outcome might be *"a model that predicts cancer treatment effectiveness per patient, achieving X% accuracy in identifying responders, and highlighting which genomic features drive the predictions."* In the Kaggle personalized medicine example, the expected outcome would be a classifier that can automatically categorize genetic variants into clinical effect categories, matching clinician annotations [13]. More generally, the project yields: - A trained model (and possibly a simple **software prototype**, like a script where a user inputs patient data and gets a recommendation). - Insights into which features (genes, lab results) were most influential, supporting the idea of personalized decision-making. - Documentation of performance on validation data (e.g., *the model's AUC was 0.85 for predicting drug response*), and discussion of how this could translate to clinical settings. The outcome should help students understand the potential and current limitations of AI in personalized medicine – for instance, recognizing that while models can find patterns in retrospect, prospective validation and interpretability are vital before real-world use.

3. Cybersecurity – Deep Learning for Anomaly Detection in Network Traffic

Objective: Apply deep learning to **detect unusual network traffic patterns** as a means of identifying cyber threats (intrusion detection). The goal is to build an Intrusion Detection System (IDS) that monitors network data (packet flows, connections, etc.) and flags deviations from normal behavior that could indicate attacks, such as malware communication, distributed denial-of-service (DDoS) traffic, or unauthorized access. By learning the typical patterns of network traffic, the model can recognize anomalies in real time – for example, a sudden spike in TCP connections or unusual packet payload signatures – and alert security teams. This project focuses on *anomaly-based detection*, which is well-suited to catching novel attacks by looking for abnormal patterns rather than known signatures [14] [15].

Key Challenges: - *Class Imbalance:* In real network data, attacks are extremely rare compared to normal traffic. This imbalance makes model training difficult – a naive model can achieve >99% accuracy by always predicting "normal" yet miss all attacks. Techniques like oversampling, synthetic data (e.g. SMOTE), or cost-sensitive learning are needed. - *Evolving Threats:* Attack patterns change as attackers adapt. The model must generalize and possibly be updated regularly. There's a risk of the model becoming outdated if trained only on old attack data. - *False Positives:* An anomaly detector should minimize false alarms. High false positives can overwhelm security teams or cause unnecessary disruptions (blocking legitimate traffic). Thus, precision in alerts is as important as catching all true attacks (recall).

Publicly Available Dataset(s): Classic datasets for network intrusion detection include: - **NSL-KDD:** An improved version of the KDD 1999 Cup dataset, containing simulated network connections labeled as normal or various attack types. While older, it's widely used for benchmarking IDS models. NSL-KDD has about 40 features per connection (duration, protocol, byte counts, etc.) and includes attack categories like DoS, probing, R2L, U2R. It is imbalanced and challenging; for instance, deep learning models on NSL-KDD report test accuracies around 82–85% and F1 in the low 80s [16] [17]. - **CIC-IDS 2017:** A more recent, realistic dataset from the Canadian Institute for Cybersecurity, containing PCAPs and extracted features from genuine benign traffic and modern attacks (e.g. Botnets, Web attacks). It's large and commonly used for evaluating deep learning IDS in contemporary settings. - **UNSW-NB15:** Another modern dataset with realistic traffic and diverse attack types, created by the Australian Cyber Security Centre. It provides rich features and is often used alongside or instead of KDD. For example, experiments report ~91% accuracy using deep networks on UNSW-NB15 [18]. These datasets are publicly available and come with predefined training/test splits and features, making them suitable for students to develop and evaluate models.

Methods and Tools: The problem can be approached either as supervised classification (if using labeled attack data) or unsupervised anomaly detection: - *Deep Learning Models:* Recurrent Neural Networks (RNNs) or LSTMs can model sequences of network events, useful for detecting temporal patterns (e.g., slow port scans). Alternatively, **autoencoders** (unsupervised neural networks) can learn to reconstruct "normal" traffic and flag anomalies when reconstruction error is high. A popular approach is using deep autoencoders or one-class neural networks to model normality. - *If treated as classification:* use deep **feed-forward neural networks** or **Convolutional Neural Networks (CNN)** on the tabular features. Some works use 1D CNN on traffic flow features to capture local patterns in feature space. Others use ensemble methods like Random Forest as a baseline. - *Tooling:* **TensorFlow/Keras** or **PyTorch** for building and training neural networks. These provide layers for dense networks, LSTMs, etc., making it straightforward to define an architecture for the IDS. Libraries such as **scikit-learn** can help with data preprocessing (standardization of features)

and provide algorithms for baseline comparisons (like Isolation Forest or SVM for anomaly detection). - Additionally, domain-specific tools: **Wireshark** or **Bro/Zeek** can be used to understand and extract network traffic features if working from raw PCAPs. However, the provided datasets already include structured features. For handling large volumes of data, one might use **Apache Spark** or Dask, but for a prototype, Python with pandas is typically enough. - *Other ideas in cybersecurity:* While network anomaly detection is the core, similar deep learning ideas apply to **malware detection** (using DL on executables or system calls) and **phishing detection** (using NLP on emails or URLs). For instance, convolutional networks have achieved >99% accuracy on certain malware classification tasks [19] [20]. Students could be encouraged to explore these extensions once the main project is done.

Evaluation Metrics: Intrusion detection has its own emphasis on certain metrics: - **Detection Rate (Recall):** The proportion of actual attacks that are correctly detected. High recall is critical (missed attacks can be disastrous). - **False Positive Rate:** The proportion of normal traffic incorrectly flagged. This should be kept low to avoid alert fatigue. Often results are reported in terms of *false alarms per hour* or similar. - **Precision:** Of the alerts raised, how many are actually attacks. Precision and recall often trade off; a balance is summarized by the **F1-score**, which is useful for overall performance, especially in imbalanced contexts. - **Accuracy** is less informative here due to class imbalance (e.g., 99.9% normal vs 0.1% attacks). A model could be >99% accurate yet miss all attacks. Therefore, **ROC-AUC** or **Precision-Recall AUC** is preferred to evaluate the model's ability to distinguish classes across thresholds [21]. - In research papers, **macro-averaged F1** and **per-class accuracy** (for each attack type) are reported. For example, one study using a deep autoencoder + Random Forest on NSL-KDD achieved ~99.8% precision and recall for overall detection, but noted lower recall on certain attack subtypes [22]. It's important to analyze such per-attack performance – some models struggle with stealthy attacks (like U2R) more than loud ones (like DoS). - If using unsupervised methods with no labels, metrics like **Mean Time To Detect** or simply the ability to detect known attack instances (treating it as an outlier detection problem and measuring true/false positive rates at a given threshold) can be used.

Expected Outcomes: By the end, students will deliver a **trained deep learning IDS model** that can ingest network feature data and output an anomaly score or binary classification (attack vs normal). For instance, the outcome could be: *"A deep autoencoder-based detector that catches 95% of network intrusions (recall) with a false positive rate below 1% on the CIC-IDS2017 test set."* Accompanying this, they should provide confusion matrices or ROC curves demonstrating performance. The project also yields: - Insight into what types of attacks were easily detected vs which were missed. For example, maybe the model catches volumetric DoS attacks well but struggles on subtle social engineering traffic; such findings can be discussed. - A discussion of model tuning for imbalance (perhaps they implemented oversampling or threshold adjustment to improve precision/recall balance). - Potential *additional ideas:* The report might mention that this anomaly detector approach could extend to other cybersecurity areas, like using a similar LSTM model on system logs for detecting insider threats, or classifying executables as malicious vs benign using deep learning. By covering "a few other ideas," the students show awareness that deep learning in cybersecurity is broad – e.g., CNNs can classify malware by binary byte patterns with high accuracy [19]. Overall, the expected outcome is a demonstration that **deep learning methods can significantly improve intrusion detection**, capable of learning complex patterns in network traffic that rule-based systems might miss [23] [24]. The project will help students understand both the power and the practical hurdles (like handling false positives) in applying AI to cybersecurity.

4. Finance – Credit Card Fraud Detection

Objective: Develop a machine learning model to detect **credit card fraud** – identifying fraudulent transactions from a large pool of legitimate transactions. Credit card companies need to flag unauthorized or suspicious charges in real time to prevent losses. The project's goal is to train a classifier on transaction data (which may include features like transaction amount, time, location, merchant category, etc.) and accurately predict whether a given transaction is fraudulent or not. The key challenge is that fraud cases are extremely rare and can evolve as fraudsters change tactics. A successful model could be deployed to alert or block likely fraudulent transactions, protecting both the customer and the financial institution.

Key Challenges: - *Severe Class Imbalance:* Typically **<0.5%** of transactions are fraudulent [25]. The model must be sensitive enough to catch those few positives while not being overwhelmed by the majority class. - *Cost of Errors:* False negatives (missed frauds) can result in direct monetary loss and chargebacks, whereas false positives (legitimate transactions flagged) harm customer experience. The cost asymmetry means recall vs precision must be balanced thoughtfully, possibly using a cost matrix or focusing on minimizing financial risk rather than raw accuracy. - *Data Privacy and Feature Engineering:* Transaction data may be sensitive (contains personal identifiers) – a project might need to use anonymized features. Also, many features might be highly correlated or require scaling; feature engineering (like creating features for transaction velocity – number of transactions in last hour – or comparing to customer's typical behavior) can significantly enhance detection but requires domain knowledge.

Publicly Available Dataset(s): A popular dataset for this task is the **Credit Card Fraud Detection dataset** released by researchers from Université Libre de Bruxelles (available on Kaggle and other platforms). It contains European card transactions from September 2013, with **284,807 transactions of which 492 are frauds (0.172% fraud rate)** [25]. All features are numerical (most are PCA-transformed for confidentiality, except features like Amount and Time). Despite the anonymization, it is widely used for evaluating fraud detection models. Another resource is the **IEEE-CIS Fraud Detection** dataset (from a Kaggle competition), which includes identity and transaction features for online card transactions, though it's more complex. For simplicity, the ULB dataset is ideal – it is highly imbalanced and hence a realistic testbed for techniques like stratified sampling, SMOTE, and anomaly detection algorithms. Public datasets allow measuring model performance in a reproducible way and comparing with published results.

Methods and Tools: Fraud detection often benefits from an ensemble of techniques: - *Machine Learning Models:* **Gradient boosting trees** (XGBoost, LightGBM) have been very effective on structured transaction data due to their ability to handle heterogeneity and highlight important features. Likewise, **logistic regression** with appropriate regularization can serve as a simple interpretable baseline. - *Deep Learning:* For large-scale data, **neural networks** (fully connected networks) can be used; however, due to imbalance, specialized architectures like autoencoders or **variational autoencoders (VAE)** are common. An autoencoder can be trained on only normal transactions, and any transaction that reconstructs poorly (high error) is flagged as potential fraud (an unsupervised approach). Additionally, recurrent models or temporal convolution may capture sequences of behavior for each card (if sequential data is available). - *Tools & Libraries:* **scikit-learn** is useful for many algorithms (logistic regression, tree models, evaluation metrics). The **imbalanced-learn** library provides utilities like SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic fraud examples and combat imbalance [26]. For deep learning, **TensorFlow/Keras** or **PyTorch** can build neural nets or autoencoders. Libraries such as **PyOD** (Python Outlier Detection) offer ready-made implementations of anomaly detection algorithms (Isolation Forest, One-Class SVM, autoencoders) that are applicable to fraud detection. Visualization tools like matplotlib or seaborn can help

plot detection rates or ROC curves. In practice, one might also use big data tools (Spark MLlib) if the dataset were huge, but for the provided dataset (few hundred thousand rows), Python on a single machine is sufficient.

Evaluation Metrics: With extreme imbalance, **accuracy** is not an informative metric – a trivial model that predicts "not fraud" for everything would be 99.8% accurate but useless. Instead, focus on: - **Precision and Recall:** Specifically, *Recall (True Positive Rate)* measures how many fraudulent transactions are caught (important to minimize losses), and *Precision* measures how many flagged transactions are actually fraudulent (important for efficiency, so investigators or automated systems don't waste effort on false alarms) [27]. In fraud detection, a common target is high recall while maintaining precision above a certain threshold to limit false alerts. - **F1-Score:** the harmonic mean of precision and recall, provides a single measure of model effectiveness on the minority class. It's useful for comparing models – for instance, an F1 of 0.80 might be a baseline to beat. In academic work on the ULB dataset, models often achieve F1-scores in the 0.80–0.90 range by careful tuning. - **ROC-AUC:** Area Under the ROC Curve is another metric, showing the trade-off between true positive rate and false positive rate. Given the imbalance, PR AUC (Area Under Precision-Recall Curve) can sometimes be more insightful. Nonetheless, high ROC-AUC (close to 1.0) has been reported with ensemble methods on the credit card fraud dataset [21]. - **Confusion Matrix & Derived Metrics:** It's useful to report the confusion matrix entries: true frauds detected vs missed, and false alarms vs true normals. From these, metrics like False Negative Rate (missed frauds proportion) and False Positive Rate can be computed. In many deployments, a threshold is set to keep false positives below a certain rate (e.g., 1 in 1000 legitimate transactions incorrectly flagged). - Given that fraud has financial cost, some evaluations use a **cost-based metric**: assign a dollar cost to false negatives (the fraud loss) and false positives (the investigation cost), then calculate the total cost for the model. This ensures the chosen model is optimal in business terms, not just statistical metrics.

Expected Outcomes: The project will result in a **fraud detection model** that can identify fraudulent transactions with a high degree of efficacy. For example, students might report: *"Our best model (XGBoost) achieved 0.95 ROC-AUC. At the chosen operating point, it catches 90% of fraudulent transactions (recall) while keeping false alarms to 2% of all transactions (precision ~30%)."* They should also demonstrate how techniques like oversampling or threshold adjustment improved the detection of the minority class [28]. The expected deliverables include: - A **comparison of algorithms**: e.g., logistic regression vs. random forest vs. neural network on the dataset, showing precision/recall/F1 for each. Often, tree ensembles perform very well; students may find something like a Random Forest gave 98% recall but needed tuning to raise precision. - **Learning about feature importance:** Using tree model feature importances or SHAP values (if advanced), they can identify which features contribute most (though in the ULB dataset the features are anonymous due to PCA). In more feature-rich data, they might say e.g. "Feature X (e.g., transaction amount relative to customer's average) was most predictive of fraud." - A discussion of the **model deployment**: how one would use the model in real-time (possibly mentioning that in practice, models are retrained periodically as fraud patterns change). Overall, students will appreciate the peculiarities of fraud detection: that success is about finding a needle in a haystack and that evaluation must be nuanced. They will have a concrete model that, on test data, demonstrates the ability to significantly outperform random or naive detection. This project also reinforces generalizable skills in handling imbalanced data and evaluating models beyond raw accuracy [29] – an important lesson for many real-world ML tasks.

5. Insurance – Insurance Fraud Detection

Objective: Create a model to detect **insurance claim fraud**, focusing on identifying fraudulent insurance claims (e.g., in auto, health, or property insurance). Insurance fraud can include exaggerated claims, false information, or entirely fake incidents. The model's objective is to analyze claim details and predict whether a given claim is likely fraudulent, enabling insurers to target those for further investigation. This not only saves money by denying illegitimate payouts, but also acts as a deterrent against fraud. The project encompasses structured data (claim metadata) and possibly unstructured data (adjuster notes or descriptions) to flag anomalies or suspicious patterns in claims.

Key Challenges: - *Data Variety:* Insurance claims involve many features – claimant info, incident details, claim amount, potentially free-text descriptions. Combining numeric, categorical, and text data can be complex. Moreover, different types of insurance have different fraud patterns. - *Imbalanced Classes:* Like credit fraud, the majority of claims are honest. Only a small percentage are fraudulent (perhaps a few percent at most). Thus, similar issues of class imbalance arise, requiring careful model training and validation. - *Adaptive Adversaries:* Fraudsters may learn the detection rules and adjust their behavior. A model might pick up a pattern (e.g., certain phrases or high amounts) and fraudsters could then avoid those, so the model needs to generalize and focus on subtle inconsistencies that are harder to fake (or continuously update as fraudsters change tactics).

Publicly Available Dataset(s): There are a few sample datasets suitable for this project: - **Insurance Company Claim Fraud Dataset:** One example (available on Kaggle) contains 1,000–10,000 insurance claim records with 30+ features and a binary label indicating fraud or not [30]. Each row represents a claim, with features like customer age, policy details, claim amount, accident type, and a flag if the claim was fraudulent. This dataset (often titled *"insurance_claims.csv"*) is a synthesized dataset for research and includes a `fraud_reported` field (Yes/No) [31]. - **Mendeley Insurance Claims dataset:** As described in a public repository, it has various features per claim (policies, customer demographics, incident details) and a *fraud_reported* label [32]. It covers vehicular, property, and personal injury claims. For example, it might include whether a police report was filed, number of witnesses, claim amount, etc., which can be predictive (fraudulent claims often have telltale anomalies in such fields). - Additionally, **auto insurance claim** datasets from Kaggle competitions or **Allstate Claim Severity** (which is about cost, not fraud) can be repurposed by injecting some fraud labels if needed. But ideally, a dataset explicitly labeled for fraud (even if synthetic) should be used. The **Insurance Fraud** Kaggle dataset (with 38 features) is one such resource [33]. Using these datasets, students can train models and validate them, learning typical fraud patterns (for instance, perhaps claims made soon after policy inception are more likely fraudulent, etc.).

Methods and Tools: The approach is akin to other fraud detection (credit card), but with potentially richer data per instance: - *Feature Engineering:* Key in insurance. Students might create features like "claim amount vs average claim for similar incidents," or "time since policy start," or flag combinations (e.g., high claim amount with low supporting documentation) as inputs. If text descriptions of incidents are available, NLP techniques (like converting text to embeddings using TF-IDF or BERT) could be incorporated to catch inconsistencies or suspicious wording. - *Machine Learning Models:* **Gradient boosting (XGBoost/LightGBM)** is a strong choice due to the mix of categorical and numeric data and its ability to handle missing values. **Logistic regression** can be used as a baseline to see how well linear separation works. **Neural networks** (multilayer perceptron) could be tried especially if there are many features or if incorporating text/image data (like photos of damage – not given here, but possible extension). - *Tools & Libraries:* Like credit fraud, **scikit-learn** and **imbalanced-learn** will be useful. For example, oversampling fraud cases or generating

synthetic ones can improve training [26]. Python's **pandas** will handle data cleaning and merging (often needed if multiple tables: customer table, policy table, claims table). If text is included, libraries such as **NLTK** or **spaCy** for preprocessing, and **Hugging Face Transformers** or **sklearn's TfidfVectorizer** for encoding text, can be used. - If the dataset is moderate in size (thousands of records), training can be done quickly on a laptop. If it were large (hundreds of thousands), using XGBoost's efficient implementation or even Spark could be relevant, but likely not necessary for available datasets. - *Frameworks:* No specialized "fraud" frameworks are needed, but one can mention **IBM's AFM (Analytics for Fraud)** or SAS Fraud Framework as industry tools (though not open-source). In open-source, **PyOD** again provides detection algorithms and could be tried for an unsupervised angle (like training an Isolation Forest on normal claims to see if known frauds score as outliers).

Evaluation Metrics: Similar to credit card fraud, emphasize metrics that account for imbalance: - **Precision, Recall, F1:** A high recall means most fraudulent claims are caught (important for saving money), while precision means a high percentage of flagged claims are truly fraudulent (important to not waste investigators' time). If the model is used as an initial filter, high recall might be prioritized (catch all frauds for manual review) at the expense of precision; or one could target a balance. The **F1-score** provides a single measure; in practice, one might aim for an F1 that is significantly better than random or current rule-based systems. - **ROC-AUC:** A good summary if threshold is adjustable; a model with ROC-AUC near 0.9 would indicate it's ranking fraudulent vs non-fraudulent claims well. For example, one might achieve an AUC of ~0.85 on a test set, indicating decent separation of fraud vs legit. - **Confusion matrix rates:** Because investigators can only handle so many cases, often a **false positive rate** threshold is set. For instance, maybe only 5% of claims can be sent for investigation – within that budget, how many frauds can we catch? This could translate to picking a threshold where the model flags ~5% of all claims, and then measure the **Precision at 5% alert rate**, which would ideally be high (meaning most of those flagged are fraud). - If data permits, **Lift or Gain charts** can be used: e.g., the model might allow an investigator to detect fraud 5 times more efficiently than random selection, which is a concrete business metric. - **Cost savings:** If we assume an average fraudulent claim payout (say \$10,000) and a false positive investigation cost (\$500), we could compute expected savings. This isn't a typical ML metric but can be a persuasive addition to the report.

Expected Outcomes: Students will produce a **classification model** that can identify fraudulent insurance claims with a certain level of confidence. The outcome could be described as: *"Our model achieves 80% recall and 90% precision in flagging fraudulent claims, meaning it catches 4 out of 5 fake claims while only 1 in 10 flagged claims is a false alarm."* This would be a strong result indicating a practical tool for insurers. The deliverables include: - The **trained model** and its performance statistics on a test set. Possibly also a list of top predictive features (e.g., "Claims made soon after policy inception" might have an odds ratio indicating higher fraud likelihood [31]). - A brief **report or poster** that might outline a few example cases: e.g., show a real fraudulent claim from the test set and how the model's output was high for it, versus a genuine claim where the score was low – to illustrate interpretability. - Discussion of how the model could be integrated into an insurance workflow: e.g., claims scoring above a threshold would be reviewed by human fraud investigators. - By doing this project, students also become aware of fraud patterns: they might note in the report, for instance, "We observed that fraudulent claims often had unusual combinations of circumstances (e.g., expensive electronics stolen shortly after a home insurance policy was purchased)." These domain insights are a valuable outcome alongside the model. In summary, the expected result is a functioning fraud detection system blueprint, with clear performance metrics and an understanding of how to adjust the model for business needs (tuning for higher precision or recall as required). This project underscores how data science can tackle a real-world finance/insurance problem and the importance of considering class imbalance and cost of errors in model development.

6. Marketing – Content Moderation on Social Media

Objective: Build a model for **content moderation** on social media, which involves automatically identifying and filtering out content that violates guidelines – for example, hate speech, harassment, explicit content, or spam. In a marketing or community management context, this helps maintain brand-safe and user-friendly social platforms. The project will likely focus on **text-based content moderation** (though image/video moderation is another facet), creating a classifier that can label user-generated content (posts, comments) as acceptable or as containing disallowed content (e.g., toxic language, spam, etc.). The aim is to assist human moderators by flagging problematic content quickly and at scale.

Key Challenges: - *Language Nuance:* Social media language is highly colloquial, often with slang, abbreviations, or deliberate obfuscation (e.g., using symbols to evade filters). The model must understand context – for instance, distinguishing an insult from a reclaimed slur, or recognizing sarcasm. - *Evolving Content:* New forms of hate speech or terms can emerge; a static model may become outdated. There's also the challenge of **adversarial content** – users intentionally altering writing to trick automated filters (like spacing out letters of a banned word). - *Bias and Fairness:* Moderation models can inadvertently be biased (e.g., flagging innocuous statements from certain dialects as toxic). Ensuring fairness and avoiding over-censorship of certain groups is important. This often requires careful dataset curation and possibly sensitive attribute checks.

Publicly Available Dataset(s): There are several well-known datasets for toxic content detection: - **Jigsaw Toxic Comment Classification dataset:** A large collection of ~150k Wikipedia comments labeled by humans for toxicity (toxic, obscene, threat, insult, identity_hate, etc.) [34]. Each comment has binary labels for multiple categories of toxic behavior. This was used in a Kaggle challenge and is widely used for training moderation models. For example, a comment might be labeled as toxic and insulting, and the model must detect that. - **Hate Speech and Offensive Language (Davidson et al. 2017):** ~25k Twitter posts labeled as hate speech, offensive (but not hate) or neither. Useful for more fine-grained classification of hate speech. - **Facebook's Hate Speech Dataset or Twitter's Kaggle Hate Speech Challenge data:** These are smaller or proprietary, but there's **"OLID" (Offensive Language Identification Dataset)** from SemEval, or **"Toxic Spans"** dataset marking toxic portions of text. - For spam, there are classic datasets like **YouTube spam collection** or **Enron email spam**, but if focusing on social media toxicity, the Jigsaw dataset is a comprehensive starting point. It is public and has a variety of toxic contexts. Using such datasets, the model can be trained to recognize known bad phrases and also generalize to similar patterns. For example, the Jigsaw dataset provides a wide range of examples that help the model learn nuances of what human moderators considered toxic [34].

Methods and Tools: This is a **Natural Language Processing (NLP)** task, typically text classification: - *Data Preprocessing:* Clean the text (handling mentions, hashtags, URLs common in social media). Possibly use tokenization appropriate for social media (handling emojis, etc., which might carry sentiment or meaning). - *Model Choices:* Modern approaches use **Transformer-based models** like **BERT** or **RoBERTa**, which can be fine-tuned to classify text into toxic/not toxic. These models capture context and usually perform best given enough data. Simpler models include **RNNs or CNNs** on text, or even classic **bag-of-words with logistic regression** as a baseline. - *Tools & Libraries:* The **Hugging Face Transformers** library makes it easy to load a pre-trained BERT and fine-tune on a labeled dataset for toxicity. There's even a pretrained model called **"DistilBERT-base-uncased-finetuned-toxic"** (from the Jigsaw data) which could be leveraged. For data handling and tokenization, **TensorFlow** or **PyTorch** along with their text utilities can be used. If using a more classic approach, **scikit-learn** provides vectorizers (CountVectorizer or TfidfVectorizer) and classifiers

(SGDClassifier, etc.). Additionally, **spaCy** or **NLTK** can help in text preprocessing (though BERT tokenization will handle a lot by itself). - *Multilingual or Multimodal:* If considering content in multiple languages or including images, it complicates matters – one might need translation or separate models per language, and computer vision models (like CNNs or CLIP for image moderation). However, the project likely sticks to English text moderation for scope. - *Other tools:* For evaluating model outputs or handling deployment, libraries like **ONNX** (to optimize models) or simple web frameworks could be used to demo a "moderation bot". But core development will be within an NLP framework.

Evaluation Metrics: Content moderation is typically a high-stakes classification. Metrics used: - **Accuracy** is fine if classes are balanced (toxic vs non-toxic might not be exactly balanced, often non-toxic content is far more common). But given class imbalance (e.g., only ~10% of comments might be toxic), **F1-score** for the positive (toxic) class is more informative. The Jigsaw competition, for instance, used AUC or F1 as evaluation metrics. - **Precision and Recall:** These are crucial. High recall means catching most toxic content (important for user safety), high precision means not wrongly flagging normal content (important for freedom of expression and user trust). Depending on platform policy, one might favor higher precision (to avoid false flags) or higher recall (to minimize undetected harassment). Often a balance is sought, and **ROC or PR curves** can illustrate the trade-off. In practice, a threshold might be set to reach a target precision (say 95%) while maximizing recall. - **ROC-AUC / PR-AUC:** Useful summary metrics for overall model discriminative ability. Since toxic content is typically rarer, **PR-AUC (Precision-Recall AUC)** is particularly useful. - **Multi-label metrics:** If the task predicts multiple categories (toxic, insult, hate, etc.), one can evaluate each label's F1 and also a combined score. For example, a macro-averaged F1 over all toxic categories can be reported. - It's also valuable to do **error analysis**: look at false positives (what clean content was mis-flagged?) and false negatives (toxic content missed). This qualitative evaluation is important in moderation to ensure the model isn't systematically biased or missing a certain kind of slur. - A special metric sometimes used is **"moderator workload reduction"** – e.g., if the model automatically removes X% of truly toxic posts with Y% precision, how much does it lighten the moderation team's job. But that can be derived from precision/recall numbers.

Expected Outcomes: The final product will be a **content moderation classifier** that can automatically flag (or score) social media text for toxicity. For example, the report might state: *"Our BERT-based model achieves 95% accuracy in identifying toxic comments, with a Precision of 0.90 and Recall of 0.85 for the 'toxic' class on the test set."* In practice, that could mean the model catches 85% of all toxic posts while only 10% of the content it flags is actually benign (which might be acceptable for a first-pass filter) [35]. Deliverables: - A confusion matrix or classification report on a test set, showing performance on toxic vs non-toxic content, and perhaps on subcategories (the model might be particularly strong at detecting insults but slightly weaker on threats, for example). - Possibly a few example outputs: e.g., show some test comments and the model's predicted label, to illustrate its capability. For instance: *"Comment: 'You are a stupid idiot.' – Model: Toxic (Insult)"*, *"Comment: 'I'll kill you!' – Model: Toxic (Threat)"*, *"Comment: 'I disagree with your point.' – Model: Non-toxic."* This builds trust that the model is doing something sensible. - The project should also discuss the **moderation pipeline**: how this model would be used (maybe it would flag content for human review, or auto-delete only the most certain cases). They might mention using a confidence threshold (e.g., only if the toxicity score > 0.98 we auto-remove, otherwise we send to human moderators). - Students will gain insight into NLP classification and also the nuanced issues of moderating content. They might note in the conclusion that certain slurs or creative insults still posed challenges, or that context matters (the word "dead" could be toxic in "drop dead!" but not in "my phone battery is dead"). They can propose improvements like using context window (conversation context) or multi-modal signals in future. In summary, the expected outcome is an effective text moderation model, evaluated with appropriate metrics,

and a clear understanding of its strengths, limitations, and how it helps social media management. Such a model could be a foundation for a safer online community by automatically filtering harmful content and thus protecting users and brand reputation.

7. Defence – Identifying Cars from Satellite Images or Drones

Objective: Develop a **computer vision model** to automatically identify (and possibly locate/count) cars in aerial images, whether from satellites or drone surveillance. This is an object detection/recognition task where the goal is to analyze high-resolution imagery and detect vehicles of interest. In defense and intelligence, such models help in monitoring traffic, detecting military equipment, or assessing activity in a region (e.g., counting vehicles in a convoy or at a facility). The project will likely focus on building an **object detection** model (e.g., using algorithms like YOLO or Faster R-CNN) that outputs bounding boxes around cars in the input images, or a simpler **classification** approach if the task is just to confirm presence of a vehicle in a given image tile.

Key Challenges: - *Small Object Detection:* In satellite imagery (e.g., 0.3 meter per pixel), cars appear very small (few pixels). Detecting small objects against complex backgrounds is challenging – the model needs to capture fine details. Drones can get closer images, but then viewpoint and scale vary. - *Variability:* Different environments (urban vs rural) have different backgrounds; vehicles can appear in many orientations, partially occluded, or in shadows. Also, vehicles come in various shapes/sizes (cars, trucks, buses) – deciding which are "cars" to detect might need definition. - *Data and Annotation:* High-quality aerial image datasets are large and labeling them (drawing boxes around every car) is labor-intensive. We rely on available datasets with annotations. There's also the challenge of **class imbalance** if images contain many more background patches than car pixels – training must handle that.

Publicly Available Dataset(s): A few notable datasets for overhead vehicle detection: - **COWC (Cars Overhead With Context):** A dataset explicitly for car detection in aerial images, containing ~32,000 labeled cars across 6 different geographic locations [36]. It provides images and annotations (often as points or bounding boxes for each car). The imagery is at high resolution (~15 cm GSD, then often downsampled to 30 cm) and includes diverse scenes (Toronto, Selwyn NZ, Potsdam & Vaihingen in Germany, Columbus & Utah in US) [36]. This diversity helps a model generalize to various backgrounds. COWC can be used for counting or detection; labels are often a single point per car (centroid). - **xView:** A very large satellite imagery dataset with objects in 60 classes, including **"Small Car" and "Truck"** classes. It has 1 million object instances labeled, making it one of the biggest aerial detection datasets [37]. In xView, cars and vehicles are annotated with bounding boxes in 0.3m resolution images. It's great for training robust detectors due to scale, though focusing on just car classes is possible. For example, xView contains ~316,000 building instances and many vehicle instances [38] [39], but one can filter to vehicle classes. - **DOTA (Dataset of Object Detection in Aerial Images):** Another large dataset with several object categories (including vehicles, like cars, trucks, etc.) in aerial images at varying angles. It provides oriented bounding boxes (since objects are rotated). - **UAVDT (Unmanned Aerial Vehicle Dataset):** Focused on vehicles in drone videos (with challenges like motion blur). For this project, **COWC** or a subset of **xView** would be a solid choice. For instance, using COWC: it might involve training on a few cities and testing on another to measure generalization. COWC has both positive (car) and negative (no car) image chips for classification tasks too [40]. The availability of these datasets means the model can be trained and quantitatively evaluated (e.g., mean Average Precision) on a standard benchmark.

Methods and Tools: This is an object detection problem: - *Model Architecture:* State-of-the-art detectors like **YOLOv5/YOLOv8 (You Only Look Once)**, **Faster R-CNN**, or **SSD (Single Shot MultiBox Detector)** are suitable. YOLO is popular for its speed and accuracy, even on smaller objects when tuned. Faster R-CNN is a two-stage detector that might achieve high accuracy but can be slower. Given small object size, using higher resolution input and perhaps tiling the image into smaller sections for processing could be necessary (since small objects can be missed if image is resized too much). - *Transfer Learning:* Often these models are pre-trained on general images (like COCO dataset). Fine-tuning on aerial data helps, but one might also use models pre-trained on similar tasks (some YOLO models fine-tuned on aerial data are available). - *Tools & Frameworks:* **TensorFlow Object Detection API** provides implementations of many detection models (including Faster R-CNN, SSD) and means to train on custom data. **PyTorch** with libraries like **Detectron2** (from Facebook) or **Ultralytics' YOLO** repository are user-friendly options to train detectors. For example, Ultralytics YOLOv5 allows training by just providing data in a certain format. - The project may involve using **OpenCV** for image processing tasks (like tiling images, drawing detection boxes on output for visualization). To manage large image files, libraries like **rasterio** (for geospatial images) or PIL can be used. - If one simplifies the task to classification (e.g., identify if a patch of image contains a car or not), a CNN like ResNet can be used. But likely detection is expected as it's more instructive. - *Techniques:* Data augmentation is important (rotations, flips) because aerial images have arbitrary orientations. Also, since car and background class imbalance is huge (most pixels are not cars), training might require hard negative mining or appropriate anchor box settings for detectors to focus on small objects. In practice, tiling a large image into smaller windows (e.g., 300x300 or 500x500 pixels) before detection can make the task easier for the model [41], as evidenced by improvements in mAP when using smaller "chips" [41].

Evaluation Metrics: In object detection, the standard metric is **mAP (mean Average Precision)**. Typically: - **Mean Average Precision at IoU=0.5 (mAP@0.5):** Measures detection accuracy – a detection is correct if it overlaps with the ground truth by at least 50% (IoU threshold) and is the correct class. mAP is the mean of average precisions for each class (but if we only care about the "car" class, AP for cars might be the focus). High mAP indicates the model is both precise and has good recall in finding objects [42]. - For example, in initial baseline results on xView, a vanilla SSD model got an mAP of ~0.145 (very low) due to difficulty of small objects, but improved multi-scale training raised it to ~0.259 [43]. These numbers illustrate the challenge – even 0.25 mAP leaves lots of missed objects [42]. - **Precision and Recall:** Besides mAP, one can report precision and recall at a certain confidence threshold. For instance, "The model achieves 80% precision and 70% recall in car detection on the test set," which is easier to interpret than mAP for some. - **Counting accuracy:** If the task includes counting cars in an area, one could evaluate the error in count (e.g., mean absolute error of the count compared to ground truth count). - **Localization accuracy:** Although mAP covers this via IoU, sometimes one might also report the distribution of IoU or how many detections were slightly off. But usually, mAP suffices. - Because multiple objects are in images, a **confusion matrix** is less standard (it's more about correct detections vs misses vs false alarms). However, one could treat detection as binary classification of each possible region and use F1, etc., but that's essentially what AP captures. - If using an alternate approach like classification of image patches (car vs no-car), then accuracy, precision, recall on that classification would be metrics (but likely object detection metrics are expected).

Expected Outcomes: The outcome will be a trained model capable of scanning aerial images and marking where cars are. Concretely: - Visual results: Students should show some example images with **bounding boxes** drawn around detected cars, demonstrating the model in action (e.g., an input satellite photo of a parking lot with the model highlighting each vehicle). This is very illustrative. - Quantitative results: e.g., *"On the test set from the COWC dataset, our model detected 90% of cars (recall) with an average precision of 0.85 at IoU 0.5."* or *"The detector achieves an AP of 0.80 for the 'car' class, meaning it's quite accurate in pinpointing*

vehicles." For perspective, models in literature on similar tasks might achieve APs in the 0.7–0.9 range for vehicles when well-trained. - If counting is the goal (say counting vehicles in a region), one could report something like *"on average, the model's count was within 5% of the true vehicle count per image."* But detection metrics are more likely. - The project will teach how to handle high-resolution imagery – possibly the need to slice images or adjust anchor boxes for small objects. The report might mention any **post-processing** used, such as merging close detections or eliminating obviously erroneous detections (perhaps filtering by size or aspect ratio). - Another expected component is addressing false detections: e.g., the model might confuse cars with other objects of similar size (like sheds or bright spots). Students can discuss how often that happened and whether additional training data or a refined model could fix it. In summary, the project yields a functioning **vehicle detection system for aerial imagery**, demonstrated on real data. This has defense applications like monitoring enemy vehicle movement or civilian applications like estimating parking lot usage. The students will have gained experience in advanced computer vision techniques and understand metrics like mAP which are standard in object detection evaluations [44]. The deliverables include the model, evaluation scores, and example detection visuals, giving a clear picture of what outcomes to expect when deploying such a model in practice.

8. Legal – Document Classification

Objective: Create a model to perform **legal document classification**, which involves automatically categorizing legal documents into predefined categories. This could mean classifying court case texts by their area of law (e.g., criminal, civil rights, tax law), labeling contracts by type (NDA, lease, employment contract), or determining the relevant law or statute that applies to a case. The goal is to assist lawyers or legal researchers by quickly organizing and retrieving documents. For example, given the text of a legal case judgment, the model might predict which articles of law are violated, or which broad topic the case falls under. This saves time in legal research and document management by triaging documents to the right experts or research folders.

Key Challenges: - *Complex Language:* Legal documents are often long, verbose, and use domain-specific jargon and syntax. Understanding the subtle meaning can be challenging for an ML model (and even for lay readers). - *Multi-label Nature:* Legal texts often pertain to multiple issues or statutes. A single case might involve both constitutional law and civil rights issues, for instance. So classification can be multi-label (multiple correct categories per document) rather than just one exclusive class. - *Data Availability and Confidentiality:* While there are public legal documents (court opinions, statutes), a lot of legal work is on private contracts or filings that aren't public. Public datasets exist but may be limited in scope. Also, models trained on one jurisdiction's data may not generalize to another due to different legal terminology.

Publicly Available Dataset(s): Several datasets and benchmarks in legal NLP are available: - **LexGLUE Benchmark:** A collection of legal NLP datasets [45]. For classification tasks, it includes: - *EUROLEX (Eur-Lex 57K):* ~65,000 EU legal documents (mostly directives and regulations) annotated with ~4,000 **concept labels from EuroVoc thesaurus** [46]. Each document can have multiple labels (topics like agriculture, trade, environment). It's a multi-label dataset. A model must output a set of relevant concepts for each document. - *SCOTUS:* U.S. Supreme Court cases mapped to one of 14 issue areas (e.g., Criminal Procedure, Civil Rights, Economic Activity) [47]. ~8,000 cases in the dataset. This is a single-label classification (each case has a primary issue area). - *ECtHR (European Court of Human Rights) Cases:* classified by articles of the Human Rights Convention that were allegedly violated [48]. That's also multi-label (a case can violate multiple articles). - *LEDGAR:* Legal clauses categorized into types (contract provisions) – a finer-grained but interesting dataset of ~100 clause types [49]. - **CaseLaw datasets:** E.g., *"CUAD – Contract Understanding Atticus Dataset"*

is for QA, not classification. But *"SEC filings classification"* or other smaller sets exist. For this project, **Eur-Lex 57K** is a prime candidate (large and multi-label), or SCOTUS (smaller and simpler single-label). Suppose we choose Eur-Lex: each document (an EU law) is labeled with several concepts. It's publicly available and has been used to benchmark multi-label text classification in the legal domain [50].

Methods and Tools: This is an NLP text classification problem with possibly very long documents: - *Text Representation:* Techniques include using **TF-IDF vectors** on the raw text (which could be extremely high-dimensional given legal vocabulary), or more advanced **Transformer models** like **Legal-BERT**. Notably, **Legal-BERT** or **CaseLaw-BERT** are language models pre-trained on legal corpora (cases, legislation) and often perform better than generic BERT on legal tasks [51]. Using such a model and fine-tuning it on the specific classification task is likely the state-of-the-art approach. - If documents are very long (some legal cases can be dozens of pages), one might need to truncate or segment input for transformer models due to token limits (512 or 1024 tokens). Summarizing or extracting key sections (like the "Holding" of a case) could help, but that adds complexity. - *Algorithms:* Apart from BERT-based models, one could try classical algorithms: e.g., **Logistic Regression** or **Linear SVM** with an n-gram TF-IDF representation. These often provide a strong baseline for text classification and can be quite effective for multi-label tasks when combined with one-vs-rest classification. - *Multi-label classification:* If multi-label, ensure the approach can output multiple labels. Algorithms like One-vs-Rest logistic regression or neural networks with a sigmoid output layer (and binary cross-entropy loss per label) are used. - *Tools:* **Hugging Face Transformers** (with models like `bert-base-uncased` or domain-specific models like `nlpaueb/legal-bert-base-uncased`) will simplify fine-tuning a transformer on a text classification task. For multi-label, one would adjust the final layer and use appropriate loss. HuggingFace's `Trainer` can handle multi-label classification by specifying the problem as multi-label. - **Scikit-learn** can be used for the classic approach (with `Pipeline` to do TF-IDF then LogisticRegression, and perhaps `MultiLabelBinarizer` to handle multi-label). - If handling large text, perhaps chunking text and using something like **Longformer** (a transformer for long documents) could be considered, but that's advanced. - Possibly **spaCy** for basic NLP preprocessing if needed (though modern models often just take raw text). - *Ontology or Keyword approaches:* In legal context, sometimes rule-based classification using keyword dictionaries or citation analysis is used (e.g., if a case cites certain statutes, that hints the category). The project might mention these, but focus on ML approaches for learning generalizable patterns.

Evaluation Metrics: Different metrics for single-label vs multi-label: - *Single-label:* Use **accuracy** and **macro-averaged F1** (especially if classes are imbalanced in frequency). For SCOTUS 14-category classification, accuracy is straightforward (e.g., a model might get ~75% accuracy in predicting the issue area). Macro-F1 ensures performance is decent across all classes, not just dominated by few large classes. - *Multi-label:* Use **micro-averaged F1** and **macro-averaged F1**. Micro-F1 (which in a multi-label context is essentially the global precision/recall considering each label assignment as separate decision) is a good overall measure especially if many possible labels [52]. Macro-F1 would emphasize performance on rare labels equally as common ones. For example, in Eur-Lex with 4k labels, one might primarily report micro-F1 (since macro-F1 could be very low due to many infrequent labels). - **Precision/Recall@k:** In some legal retrieval contexts, they measure if the correct labels are in the top k predicted labels. But F1 covers similar ground. - If focusing on a multi-class single-label, a **confusion matrix** can show which categories get confused (e.g., model mixing up "Property Law" vs "Contracts" cases). - In LexGLUE evaluations, models often use **exact match** (for multi-label, did it predict the exact set) which is very strict, and macro/micro F1 which are more forgiving [53]. F1 is likely the primary metric of interest for multi-label [52]. - Possibly **subset accuracy** for multi-label (exact match accuracy), but with many labels that's usually very low and not so informative (except for tasks with fixed small label sets). - **Example-based metrics:** e.g., average precision per

document or Jaccard similarity between predicted and true label sets. But these are less common than F1 in literature.

Expected Outcomes: The project will produce a **trained classifier** that can label legal documents appropriately. For instance: *"Our Legal-BERT model achieves a micro-F1 of 0.80 on the Eur-Lex dataset, effectively tagging EU laws with the relevant concepts."* or *"The model classifies Supreme Court opinions by issue area with 85% accuracy."* These figures would be compared to baselines or known benchmarks (for context, a baseline BERT model might achieve ~78% micro-F1 on Eur-Lex, whereas a Legal-BERT could push >80% [51]). - The deliverable includes the model's performance metrics and perhaps some **example outputs**: e.g., show a snippet of a document and the labels the model predicted versus the gold labels. For instance: a case text summary and "Predicted: {Freedom of Expression, Privacy}; Actual: {Privacy, Fair Trial}" to analyze mistakes (maybe it missed one label). - If multiple approaches were tried (say TF-IDF vs BERT), the report can compare them. Often, BERT will outperform, but the simpler model might be faster – students could mention trade-offs. - **Interpretability:** Legal professionals might want to know *why* a document was classified a certain way. While not required, students might use techniques like looking at the top words for each class in a linear model, or using LIME/SHAP for a transformer to identify key phrases that influenced a classification. For example, if a case is classified under "Patent Law," the presence of words like "patent," "invention," "USPTO" might be the reason – which is intuitive. Highlighting such evidence increases trust in the model. - The project demonstrates how AI can **assist legal workflows**: e.g., automatically routing new case filings to the right department or suggesting relevant past cases by topic. The expected outcome description could envision: *"Given a new legal document, our system can instantly suggest relevant tags (like applicable law provisions or legal topic), which can accelerate document filing or retrieval in a law firm."* - Additionally, students will note limitations: legal text can be subtle; the model might misclassify if two areas of law share vocabulary, or if the text is too long (leading to truncation issues). They might mention that combining this with a summarization model could further improve it (as a forward-looking idea). In essence, the outcome is a practical tool for legal document management – achieving good accuracy on a challenging NLP task – and a structured report that includes the problem definition, dataset used, choice of model, evaluation results (with citations to any benchmark for context), and examples of model predictions to illustrate its behavior. This will help students learn about adapting NLP to a specialized domain and the importance of domain-specific data or models [54] [45].

---

[1] [2] [3] [4] [5] Privacy-first health research with federated learning | npj Digital Medicine
https://www.nature.com/articles/s41746-021-00489-2?error=cookies_not_supported&code=6eb04366-ff31-4699-ada7-8e4c4056e615

[6] Knowledge abstraction and filtering based federated learning over heterogeneous data views in healthcare | npj Digital Medicine
https://www.nature.com/articles/s41746-024-01272-9?error=cookies_not_supported&code=7d0c7f0a-0171-4035-8f76-d11281f3415a

[7] Revolutionizing healthcare data analytics with federated learning
https://pmc.ncbi.nlm.nih.gov/articles/PMC12213103/

[8] TensorFlow Federated
https://www.tensorflow.org/federated

[9] Top 7 Open-Source Frameworks for Federated Learning - Apheris
https://www.apheris.com/resources/blog/top-7-open-source-frameworks-for-federated-learning

10 Flower: A Friendly Federated AI Framework
https://flower.ai/

11 12 13 Personalized Medicine Kaggle Competition
https://philippschw.github.io/machine_learning/classification/healthcare/personalized-medicine/

14 15 17 18 23 24 A Deep Learning Approach for Network Intrusion Detection Using a Small Features Vector
https://www.mdpi.com/2624-800X/3/3/23

16 A deep learning technique for intrusion detection system using a ...
https://www.sciencedirect.com/science/article/pii/S0140366422004601

19 Malware detection using deep learning : r/cybersecurity - Reddit
https://www.reddit.com/r/cybersecurity/comments/1f3h47w/malware_detection_using_deep_learning/

20 Automated machine learning for deep learning based malware ...
https://www.sciencedirect.com/science/article/abs/pii/S0167404823004923

21 Optimizing credit card fraud detection with random forests and SMOTE
https://pmc.ncbi.nlm.nih.gov/articles/PMC12098799/

22 Optimization of predictive performance of intrusion detection system ...
https://pmc.ncbi.nlm.nih.gov/articles/PMC10496009/

25 Credit Card Fraud Detection
https://zenodo.org/records/7395559

26 28 30 31 32 insurance_claims - Mendeley Data
https://data.mendeley.com/datasets/992mh7dk9y/2

27 Credit Fraud || Dealing with Imbalanced Datasets - Kaggle
https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets

29 A Critical Examination of Credit Card Fraud Detection Methodologies
https://arxiv.org/html/2506.02703v1

33 Insurance Fraud Detection - Kaggle
https://www.kaggle.com/datasets/arpan129/insurance-fraud-detection

34 google/jigsaw_toxicity_pred · Datasets at Hugging Face
https://huggingface.co/datasets/google/jigsaw_toxicity_pred

35 Content moderation - Anthropic API
https://docs.anthropic.com/en/docs/about-claude/use-case-guides/content-moderation

36 Car Localization and Counting with Overhead Imagery, an Interactive Exploration | by Adam Van Etten | The DownLinQ | Medium
https://medium.com/the-downlinq/car-localization-and-counting-with-overhead-imagery-an-interactive-exploration-9d5a029a596b

37 38 39 41 42 43 44 The XView Dataset and Baseline Results | by Roger Fong | Picterra | Medium
https://medium.com/picterra/the-xview-dataset-and-baseline-results-5ab4a1d0f47f

40 Cars Overhead With Context Dataset at LLNL
https://gdo152.llnl.gov/cowc/

45 46 47 48 49 53 54 Open Source Legal: LexGLUE
https://opensource.legal/projects/LexGLUE

50 51 Computational Law: Datasets, Benchmarks, and Ontologies - arXiv
https://arxiv.org/html/2503.04305v1

52 aclanthology.org
https://aclanthology.org/2022.acl-long.297.pdf