LINEAR PROGRAMMING-BASED CLASSIFIER FOR BREAST CANCER DIAGNOSIS

KHANH DINH, RAMISA MOZUMDAR, BELLA DELMONTE
DEPARTMENT OF MATHEMATICS AND STATISTICS, MOUNT HOLYOKE COLLEGE

1 Executive Summary

Motivation. Accurate breast cancer diagnosis is critical for effective treatment and patient outcomes, as misdiagnoses can lead to unnecessary interventions or delayed treatments. While current diagnostic methods (e.g. imaging, tissue examination) are highly effective, there is merit in developing computational tools to assist clinicians in interpreting diagnostic data. This project explores a tumor classification method based on linear programming, designed to enhance diagnostic reliability by minimizing misclassification.

Data. This project uses the Wisconsin Diagnostic Breast Cancer (WDBC) dataset [6], which contains 569 samples, each corresponding to a fine needle aspirate (FNA) of a breast mass. For each breast mass, 30 numerical variables describe characteristics of the cell nucleus. Diagnoses for each breast mass are labeled either malignant or benign.

Modeling strategy. We formulated tumor classification as a linear program (LP) that seeks to determine a hyperplane $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}+b$ that maximally separates malignant and benign breast masses. Our LP model introduces slack variables ξ_i that measure margin violations, with the objective of minimizing their sum to minimize misclassifications. Following the method of Bennett and Mangasarian [4], we sought an optimal separating hyperplane by solving the following LP:

$$\min_{\mathbf{w},b,\xi} \sum_{i=1}^{n} \xi_i \quad \text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) + \xi_i \geq 1, \ \xi_i \geq 0.$$

We implemented the model using five predictive features from the WDBC dataset – radius_mean, perimeter_mean, area_mean, concavity_mean, and concave points_mean – based on the recommendation from [1] and to reduce model complexity. The true diagnosis labels were encoded as $+1$ for malignant and -1 for benign. We solved this LP using `scipy.optimize.linprog` (HiGHS), which produced optimal $\mathbf{w}$, b, and ξ.

Results. The classifier achieved a 91.7% training accuracy on the entire dataset, misclassifying 47 tumors out of 569, with a specificity of 95.0% and a sensitivity of 86.3%. The optimal hyperplane had weight vector $\mathbf{w} = [-5.84, 0.53, 7.70, 0.01, 1.81]$ and bias $b = 0.33$. 53 tumors were correctly classified but fell within the ± 1 margin.

Interpretation. The area_mean and concave points_mean features have the largest positive weights, indicating that larger tumor size and more irregular (concave) nuclei strongly push predictions toward malignant, consistent with medical intuition. Of the 569 tumors, 47 were misclassified and 53 were correctly but less confidently classified within the ± 1 margin. The remaining cases were confidently classified with a margin of at least 1. The resulting linear model is simple (five feature weights and one bias) and thus produces an interpretable decision boundary that could potentially be adjusted for clinical preference (e.g. avoiding false negatives). The dual formulation (discussed in the full report) reveals which cases are critical (support vectors) and how the approach could scale or be improved.

Impact. The single-plane rule is transparent and tunable (e.g., threshold shifts or cost-weighted slacks) for different clinical risk preferences. A 10-feature extension (Appendix B) reduced the slack variable sum (optimal value) to 32 and achieves 96% accuracy while retaining interpretability.

Cautions. A single hyperplane cannot capture nonlinear or multimodal class boundaries, so borderline cases may remain misclassified. The LP minimizes slack uniformly, but clinical risk is not symmetric: false negatives are realistically more consequential than false positives. A cost-weighted version would better reflect this. Correlated features may yield unstable weight signs as the LP only minimizes slack. The headline 91.7% accuracy is in-sample; generalization was inspected on a 30% hold-out split (Appendix C) but not cross-validated. Limiting to five features aids interpretability but omits potentially useful predictors (texture, smoothness, etc.).

Reproducibility. Python code for data handling and optimization is provided in Appendices A–D; running `lpsolver.py` regenerates every numeric claim in this summary.

Date: August 1, 2025.

2 Background

Breast cancer is the most frequently diagnosed cancer and the leading cause of cancer-related death among women worldwide. Prognosis depends critically on early, accurate detection; a missed malignancy can delay treatment, while a false positive can trigger unnecessary biopsies or surgery. One common diagnostic procedure is *fine-needle aspiration* (FNA), where a thin needle extracts cells from a suspicious breast mass for cytological examination. Digital imaging of the extracted nuclei yields quantitative measurements of their size, shape, and texture.

The Wisconsin Diagnostic Breast Cancer dataset (WDBC) [6] provides quantitative features extracted from images of these cell nuclei. Traditional statistical methods (like linear discriminant analysis) and modern machine learning techniques (like support vector machines and neural networks) have been applied to this problem. However, clinicians often prefer models that are interpretable and based on understandable rules, so that the decisions can be transparently explained. Linear models fulfill this need by providing a weight for each feature that indicates its influence on the prediction. Building on the linear programming discrimination work of Bennett and Mangasarian [4], we cast WDBC classification as a single LP that minimizes total misclassification slack.

In the following sections, we detail the model formulation, implementation, results on the dataset, and the implications of using an LP classifier in a medical setting.

3 Methods

3.1 Data and Preprocessing

The dataset comprises 569 instances, each with 30 numerical continuous features describing properties of cell nuclei (e.g., radius, perimeter, area, concavity) across three metrics: mean, standard error, and worst, and a diagnosis label (M = malignant, B = benign).

For this project, we focused on a subset of five features:

- radius_mean: mean distance from center to perimeter of nucleus (microns),
- perimeter_mean: mean perimeter length of nucleus,
- area_mean: mean area of the nucleus,
- concavity_mean: mean severity of concave portions of the nucleus outline (dimensionless),
- concave points_mean: mean number of concave (indentation) points on the nucleus outline.

These features were chosen because they are known to be diagnostically relevant and capture both size (radius, perimeter, area) and shape (concavity measures) characteristics of the tumor. Focusing on five features also simplifies the model and improves the tractability of the LP solver.

Before modeling, we standardized each selected feature to zero mean and unit standard deviation: $z = \frac{x-\mu}{\sigma}$ where x is a raw data value, μ is the feature mean, and σ its standard deviation. Standardization is essential due to differing feature scales (e.g., area values are much larger than concavity, preventing optimization biases and ensuring meaningful weight comparisons. This transformation makes all features dimensionless and comparable.

The target variable in the dataset is categorical (Malignant or Benign). We encoded this into a binary numeric variable $y_i \in +1, -1$ for each sample i, where $+1$ represents malignant and -1 represents benign. This encoding is convenient for the formulation of a linear classifier: a correctly classified sample i should satisfy $y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0$.

3.2 Model Formulation

For modeling and evaluation, we treated the entire dataset as our sample. In a real-world scenario, one would typically split data into training and testing sets or use cross-validation to get generalization performance. Here, however, our goal was to demonstrate the LP formulation and interpret the resulting model, so we trained on all 569 instances and evaluated the training performance (with the understanding that true generalization would need separate testing). We did verify that the optimization approach could handle the full dataset. Initially, for development, we limited the solver to a subset of 100 samples to ensure tractability, but with the modern `HiGHS` LP solver in SciPy, the full dataset (569 samples, 5 features) was solved efficiently in a fraction of a second. All reported results use the full dataset. Experiments with training/testing splits are considered in Appendix C.

We formulated the classification as a linear programming problem in the following way. We seek a linear decision function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ (where $\mathbf{w}$ is a weight vector of length 5 for the features and b is a scalar bias or intercept) such that $f(\mathbf{x})$ is positive for malignant cases and negative for benign cases. In an ideal perfectly separable scenario, we could enforce $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ for all samples i by appropriately scaling $\mathbf{w}$ and b. This inequality means that each sample is not only correctly classified (sign of the left side positive) but also lies at least at a distance (margin) of 1 from the decision boundary $f(\mathbf{x}) = 0$. The value 1 is an arbitrary margin threshold which we include for convenience – any constant would do, since $\mathbf{w}$ and b can be scaled – but 1 is a conventional choice. The margin makes the classifier robust in that it separates classes with a "buffer zone." However, real data are often not linearly separable. Thus, we introduce slack variables $\xi_i \geq 0$ for each sample i to allow violations of the margin requirement. The slack ξ_i effectively measures how much sample i falls short of the ideal margin. The constraints with slack become:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \ldots, n$$

where $n = 569$ is the number of training samples. If $\xi_i = 0$, then $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$, meaning sample i is correctly classified with margin at least 1. If $0 < \xi_i < 1$, sample i is correctly classified ($y_i(\mathbf{w}^T\mathbf{x}_i + b)$ is still positive), but the margin is less than 1 (it's $1 - \xi_i$). If $\xi_i \geq 1$, then the left side could be zero or negative, indicating sample i is misclassified (if $y_i(\mathbf{w}^T\mathbf{x}_i + b) < 0$, the constraint would require $\xi_i > 1$ to hold). Thus, slack values $\xi_i \geq 1$ correspond to misclassifications, and ξ_i exactly equals the amount by which the sample is on the wrong side of the decision boundary (for example, $\xi_i = 1.5$ means $y_i(\mathbf{w}^T\mathbf{x}_i + b) = -0.5$, so the point is 0.5 on the wrong side). We want to find $\mathbf{w}, b$ that minimize the total violation of these constraints. A natural objective is to minimize the sum of all slack variables $\sum_{i=1}^{n} \xi_i$. This objective effectively tries to minimize the number of misclassified points, treating all margin shortfalls linearly. (In contrast, support vector machines typically minimize a weighted combination of $|\mathbf{w}|^2$ and $\sum \xi_i$, which balances margin maximization and error minimization. Here we focus solely on error minimization, which will tend to find a separating plane if one exists, without regard to maximizing the margin beyond 1.) Putting it together, our linear programming formulation is:

$$\text{Minimize} \quad \sum_{i=1}^{n} \xi_i \quad \text{subject to} \qquad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \quad \xi_i \geq 0, \quad i = 1, \ldots, n.$$

in which:

- $\mathbf{x}_i = [x_{i1}, x_{i2}, x_{i3}, x_{i4}, x_{i5}]^T$: vector of 5 features for the i^{th} sample: radius_mean, perimeter_mean, area_mean, concavity_mean, concave_points_mean
- $y_i \in \{-1, 1\}$: diagnosis label (Benign = -1, Malignant = 1)
- $\mathbf{w} = [w_1, w_2, w_3, w_4, w_5]^T$: weights
- $b \in \mathbb{R}$: bias
- $\xi_i \geq 0$: slack variables (one per data point)

This is a linear program (LP) because the objective is linear in the decision variables (the ξ_i's) and the constraints are linear in $\mathbf{w}$, b, and ξ_i. The decision variables in this LP are the components of $\mathbf{w}$ (5 variables), the bias b (1 variable), and the slack variables $\xi_1, \ldots, \xi_n$ (one for each sample, so 569 variables). In total, the LP has 5 + 1 + 569 = 575 decision variables. The number of constraints is n for the classification constraints plus n for the slack non-negativity constraints, for a total of 1138 inequality constraints. (We did not explicitly include equality constraints; the bias and weights are unconstrained in sign.) A couple of points are noteworthy about this formulation:

- There is no explicit variable or penalty for $\mathbf{w}$ in the objective. This means that if the classes are perfectly separable (so we could achieve $\xi_i = 0$ for all i), then any scalar multiple of a solution $(\mathbf{w}, b)$ is also a solution (since scaling $\mathbf{w}$ and b down will still satisfy $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ until it just meets the margin for the support points). In other words, the solution may not be unique without some normalization. In practice, if multiple solutions yield the same minimal objective, LP solvers will return one of them (often one that satisfies some internal tie-breaking conditions). In our case, the data are not perfectly separable, so $\sum \xi_i$ will be > 0 and the solution tends to be unique.
- The constraints $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$ are equivalent to $y_i(\mathbf{w}^T\mathbf{x}_i + b) + \xi_i \geq 1$ which can be rewritten as $y_i \cdot (w_1x_{i1} + w_2x_{i2} + w_3x_{i3} + w_4x_{i4} + w_5x_{i5} + b) + \xi_i \geq 1, \quad \xi_i \geq 0, \quad i = 1, 2, \ldots, n.$ We implement them in that form for the solver. Each such constraint essentially forces the decision function for sample i to reach 1 unless ξ_i pays the cost for falling short.

3.3 Implementation with Python

Full implementation details are provided in Appendix A; the same code can be executed to reproduce every value and figure in this paper. Appendix B shows the analogous script for a ten-feature model, while Appendix C contains the PCA visualisations, the train–test split, and the confusion-matrix graphics quoted in Section 4.

The data was loaded using the `ucimlrepo` library (which fetches datasets from the UCI repository), and we used `pandas` and `numpy` for data handling and preprocessing. After standardizing the features and encoding the labels as described, we set up the matrices for the LP constraints in the form required by SciPy's `linprog` function. The `linprog` function (from `scipy.optimize`) solves linear programming problems of the form: $\min c^T x$ subject to $A_{ub}x \leq b_{ub}$, $A_{eq}x = b_{eq}$ and bound constraints on $\mathbf{x}$. In our case, we transformed each constraint $y_i(\mathbf{w}^T\mathbf{x}i + b) + \xi_i \geq 1$ into the form $Aub\mathbf{x} \leq \mathbf{b}ub$. This is done by rearranging:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) + \xi_i \geq 1 \iff -y_i(\mathbf{w}^T\mathbf{x}_i + b) - \xi_i \leq -1.$$

We constructed each such constraint row in A_{ub} as follows:

- The columns of A_{ub} correspond to the variables in a specific order: $[w_1, \ldots, w_5, b, \xi_1, \ldots, \xi_n]$.
- For the i-th sample's constraint, we fill in the coefficients for $w_1 \ldots w_5$ as $-y_i x_{i1}, -y_i x_{i2}, \ldots, -y_i x_{i5}$ (where x_{ij} is the j-th feature value of sample i), the coefficient for b as $-y_i$, and the coefficient for ξ_i as -1. All other ξ_j for $j \neq i$ get a coefficient of 0 in this constraint.
- The right-hand side $\mathbf{b}\_ub$ for this constraint is -1.

Additionally, for each i we have a non-negativity constraint $\xi_i \geq 0$, which we also express as $-\xi_i \leq 0$ for the solver. These are straightforward: for each ξ_i, we set a row in Aub that has -1 in the column corresponding to ξ_i and 0 elsewhere, with 0 on the right-hand side. In practice, `linprog` allows us to specify variable bounds directly, so we could simply specify ξ_i bounds as $[0, \infty)$ and not add these constraints. We included them explicitly for clarity and to double-check that the solver's output respects them. The objective vector $\mathbf{c}$ is set to have 0 for each weight and for b, and 1 for each slack variable. Thus, $\mathbf{c}^T\mathbf{x} = \sum_i 1 \cdot \xi_i$, which is exactly the objective $\sum_i \xi_i$.

We used SciPy's `linprog` with the method "highs" (which is a simplex solver) to solve this optimization. The HiGHS solver is very efficient with this problem size (575 variables, 1138 constraints), finding the optimum in a negligible amount of time. We had to specify bounds for the variables: by default, `linprog` assumes all variables ≥ 0, which is fine for ξ_i but not correct for w_j or b since weights and bias can be negative. We set the bounds for $w_1, \ldots, w_5$ and b to $(-\infty, +\infty)$ (in code, `None` for no bound) and for each ξ_i to $(0, +\infty)$. Once the solver found an optimal solution, we extracted the resulting weight vector $\mathbf{w}$ and bias b from the solution vector, and the slack values ξ_i. We then evaluated how many ξ_i were zero or nonzero, how many exceeded 1, etc., to interpret the training results. We also computed the resulting classifier's predictions on the training set to calculate the training accuracy and to examine any patterns in the misclassified cases.

4 Results

After solving the linear program, we obtained an optimal linear classifier defined by a weight vector $\mathbf{w}$ for the five features and a bias b. Table 1 lists the values of the weights and bias obtained.

Feature (standardized)	Weight w_j
radius_mean	−5.8357
perimeter_mean	+0.5273
area_mean	+7.6966
concavity_mean	+0.0119
concave points_mean	+1.8091
Bias b	+0.3302

Table 1. Values of weights and bias in five-feature model.

The decision function is:

$$f(\mathbf{x}) = -5.8357 \times \text{radius\_mean} + 0.5273 \times \text{perimeter\_mean} + 7.6966 \times \text{area\_mean} + 0.0119 \times \text{concavity\_mean} + 1.8091 \times \text{concave points\_mean} + 0.3302.$$

Interpretation. area_mean has a large positive weight, meaning larger tumor cell area pushes $f(\mathbf{x})$ higher (toward predicting malignant, since malignant is encoded as +1). concave points_mean also has a positive weight, indicating that more concave points (irregularly shaped nuclei) increase the malignancy score.

Interestingly, radius_mean has a negative weight despite being typically correlated with area; this likely reflects redundancy with area and perimeter, and the solver found a combination where radius gets a negative coefficient to fine-tune the decision boundary in the correlated feature space. (Multicollinearity among radius, perimeter, and area can lead to such weight sign inversions. The model still uses a weighted combination of these to assess size.)

The weight for concavity_mean is very close to zero, suggesting that this feature did not contribute much beyond what concave points_mean captured—indeed, in the data, concavity and concave points are correlated measures of shape.

The bias b is positive 0.3302, which sets the threshold such that a sample with all features around the mean values (zero standardized) would be classified as benign if the weighted sum of features is below -0.3302 and malignant if above that. Because some weights are large, a sizable deviation in certain features is needed to flip the sign of $f(\mathbf{x})$.

Next, we examine the training classification performance. The LP objective minimized the sum of slacks $\sum_i \xi_i$, and the optimal objective value was 102.59. This number is the sum of margin shortfalls across all points. We found that out of 569 samples: 469 samples have $\xi_i = 0$. These points satisfy $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$, meaning they are correctly classified and lie outside or on the margin boundary. These can be considered well-classified with a safety margin. 53 samples have $0 < \xi_i < 1$. These points are correctly classified (since $y_i(\mathbf{w}^T\mathbf{x}_i + b)$ is still positive, albeit less than 1). They lie between the margin and the decision boundary. These are borderline cases – the classifier gets them right, but with less confidence (if we needed a margin of 1 for absolute confidence, they fall short by ξ_i). 47 samples have $\xi_i \geq 1$. These correspond to misclassified points. For these, $y_i(\mathbf{w}^T\mathbf{x}_i + b) < 0$ (the left side of their constraint), meaning the linear classifier's prediction is on the wrong side. The slack value minus 1 gives how far into the wrong side they are. For instance, the largest ξ_i among these was about 3.1, indicating one sample is significantly misclassified (the classifier output was -2.1 for that sample, since it needed $\xi = 3.1$ to satisfy $-2.1 + 3.1 \geq 1$). From the above, we can calculate the training accuracy: the classifier correctly classified $469 + 53 = 522$ samples and misclassified 47. The accuracy is $522/569 \approx 91.7\%$. This is a strong performance given that only 5 features were used. In practical terms, there were 29 malignant cases misclassified as benign (false negatives) and 18 benign cases misclassified as malignant (false positives) in the training set. The sensitivity (true positive rate for malignancy) on training data is 86.3%, and the specificity (true negative rate for benign) is 95.0%. These numbers indicate the classifier errs a bit more on missing some malignant tumors (which is something we discuss addressing in the next section).

In summary, the LP classifier achieved a high accuracy on the training data, separating the two classes with a linear boundary that mostly aligns with the intuitive direction of feature effects (large, irregular nuclei $\rightarrow$ malignant). The results match expectations for this dataset: using just a few of the strongest features, a linear model can do quite well, though it will misclassify some cases that are inherently ambiguous or outliers. Compared to a fully supervised approach with all 30 features, our model might have slightly lower accuracy, but it is far simpler and more interpretable. We next discuss the implications of these findings and how one might further analyze or improve the model.

5 Discussion

The linear programming approach provided a clear, interpretable model for breast cancer diagnosis. The resulting linear classifier can be interpreted in terms of feature importance and makes intuitive sense in the context of the medical problem. Features related to tumor size (area, perimeter) and shape irregularity (concave points) were key drivers in the model's decisions. The sign of the weights indicates the direction of influence: as expected, larger values of area or more concave points increase the likelihood of malignancy (positive weights), whereas in the model, a larger radius_mean actually decreases it (negative weight) – this counterintuitive sign is likely due to multicollinearity among the size features. In practice, one might address that by removing one of the correlated features or by adding a small regularization on the weights to prefer smaller weights (which often resolves sign instabilities). Nonetheless, the magnitude of the weights tells us that area is the most influential single feature (in standardized terms), concave points is second, and the others are comparatively less important.

One benefit of the LP formulation is that we can also examine the **dual variables** associated with the constraints to get insight into which samples were most critical. In our solution, we found that 106 of the inequality constraints had non-zero dual values (shadow prices). These correspond to 106 samples that are either on the margin or misclassified – in other words, these are the cases that the optimizer had to actively consider to pin down the hyperplane. Samples with $\xi_i = 0$ and strictly $y_i(\mathbf{w}^T\mathbf{x}_i + b) > 1$ (well outside margin) typically have zero dual; they don't influence the solution because they are sufficiently far and could be moved slightly without affecting the objective. But samples with $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$ or < 1 (margin points and errors) will generally have positive dual weights in the optimal solution. These dual values reflect how strongly each constraint is binding: a larger dual value means if we slightly relaxed that constraint (allowed a bit more violation), the objective would improve at a certain rate. Thus, points that are misclassified or exactly on the margin could be considered the "supporting" points of the classification plane. [5] In a medical context, it could be useful to flag these support points – they might be borderline cases that could need further investigation or might indicate areas where the model is less confident.

In terms of classifier performance, an accuracy of ~91.7% on training data is strong given only five features. However, we know from [2] and by our own experiment that using 10 features achieves the best accuracy of 96%. The LP classifier here is essentially a baseline linear model. Overfitting risk is actually relatively low here because we did not try to aggressively fit noise.

As an interpretable, margin-aware extension, we also implemented a hinge-loss LP—decomposing each weight into nonnegative w^{+}/w^{-} variables and applying slack penalties and solved it via SciPy's HiGHS solver in Appendix D. This approach produces a sparse, easily understood classifier with error rates on par with our baseline model.

5.1 Recommendations

Scalability-wise, solving an LP with a few hundred samples and features is very fast (almost immediately, as we experienced in Google Colab and VSCode).

From a practical standpoint, the linear model's interpretability is a major advantage, although our model definitely needs improvements. A doctor could use the linear formula to manually compute a score for a tumor (after standardizing its features) and see whether it's above or below the threshold. The weights themselves align with medical knowledge: malignant tumors generally have larger and more irregular nuclei. The negative weight on radius could prompt further analysis – it might be that when the area is already accounted for, radius (which is highly correlated) acts as a correcting factor for some cases. For example, if two tumors have equal area but one has a larger radius and lower perimeter (which could happen if the shape differs), the one with disproportionately large radius (for its area) might be more benign-looking (perhaps smoother edges), hence the model gives it a benign tilt.

One concern in deploying such a model is the cost of errors. In our results, the model had more false negatives (29) than false positives (18). In cancer diagnosis, a false negative is more serious (missing a cancer) than a false positive (which might lead to an extra biopsy). We could tune the decision threshold to reduce false negatives. Because the LP naturally gave a boundary at $f(\mathbf{x}) = 0$, we can adjust this by adding a safety bias. For instance, lowering the threshold would increase sensitivity at the expense of specificity. Alternatively, we could modify the LP to weight the slack for malignant cases higher than the slack for benign cases. This would involve changing the objective to $\min \sum_i C_i \xi_i$ where C_i is larger for a malignant sample (if misclassified) than for a benign sample.

5.2 Limitations and Future Directions

To keep the project and the report focused on the practice of linear-programming formulation, we intentionally omit engineering details such as exhaustive train–validation splits, hyper-parameter sweeps, or automated feature-subset search. A reproducible 70/30 split and the ten-feature extension are provided in Appendices C and B; readers can use those as starting points for more exhaustive benchmarking.

Any single hyperplane can only separate data that are approximately linearly separable. About 8% of WDBC samples fall on the wrong side of *every* possible plane in the five-dimensional feature space; those observations force positive slack in our objective. Using more features will yield a better separation, but might still not perfectly separate the data. Nonlinear classifiers (like kernel SVM, decision trees, neural networks) could achieve higher accuracy by capturing interactions or nonlinear patterns. However, they might lose some interpretability. An interesting middle ground is piecewise-linear models. In fact, Bennett (1992) had mentioned constructing a decision tree via linear programming. [3]

Our LP solution does not explicitly maximize the margin between classes. This means there could be many solutions with similar slack sums but different orientations, especially if there are regions with no data. In practice, the solution we found seems reasonable. If we wanted a unique or more stable solution, we could add a small term to the objective like $\epsilon|\mathbf{w}|_1$ (absolute sum of weights) which would make it a linear objective still (just more variables) and tends to sparsify $\mathbf{w}$ or at least limit extreme values.

All slacks are penalized equally, yet in practice, a false negative (missed malignancy) is far more serious than a false positive. A cost-weighted LP (simply scale ξ_i by a class-dependent constant) is straightforward and is a natural extension for clinical deployment.

All five chosen features remained in the model, but one had a negligible contribution. Restricting the model to five variables excludes informative predictors such as texture or smoothness metrics. The ten-feature extension in Appendix B reduces the total slack from 102.6 to 32.0 and cuts misclassifications from 47 to 11 while keeping the model linear and readable. Multicollinearity among size-related features (radius, perimeter, area) can still flip weight signs unpredictably.

The accuracy of 91.7% is in-sample only. A hold-out evaluation (Appendix C) shows comparable performance, but no cross-validation was performed. Our future work might incorporate k-fold cross-validation or bootstrapping.

References

[1] Breast cancer prediction — kaggle.com. `https://www.kaggle.com/code/buddhiniw/breast-cancer-prediction`. [Accessed 04-05-2025].

[2] Feature Selection and Data Visualization — kaggle.com. `https://www.kaggle.com/code/kanncaa1/feature-selection-and-data-visualization#Conclusion`. [Accessed 05-05-2025].

[3] Kristin P Bennett. Decision tree construction via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1992.

[4] Kristin P. Bennett and O. L. Mangasarian and. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1(1):23–34, 1992.

[5] David González-Patiño, Yenny Villuendas-Rey, Magdalena Saldaña-Pérez, and Amadeo-José Argüelles-Cruz. A novel bioinspired algorithm for mixed and incomplete breast cancer data classification. *International Journal of Environmental Research and Public Health*, 20(4):3240, 2023.

[6] Mangasarian-Olvi Street Nick Wolberg, William and W. Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5DW2B.

A Five-Feature LP Solver

The script below (`lpsolver.py`) retrieves the WDBC dataset, selects five features, standardizes them, formulates the linear programming classifier exactly as described in Section 2, and solves it with SciPy's HiGHS. Running this file reproduces the weight vector in Table 1 and the training performance metrics in Section 4.

```
# -*- coding: utf-8 -*-
"""LPsolver.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/11cq8NtecZz-NIivOEW9NfJo_G7e8yUnA

## Summary
This notebook demonstrates how to use linear programming (LP) to build a linear
    classifier for the Breast Cancer Wisconsin (Diagnostic) dataset. We formulate the
    classification task as a linear programming optimization problem, introducing slack
     variables to handle misclassifications.
"""

# import
!pip install ucimlrepo

import pandas as pd
import numpy as np
from ucimlrepo import fetch_ucirepo
from scipy.optimize import linprog
from sklearn.preprocessing import StandardScaler

# fetch dataset
data_raw = fetch_ucirepo(id=17)

# data (as pandas dataframes)
X_raw = data_raw.data.features
y_raw = data_raw.data.targets

# # metadata
# print(data_raw.metadata)

# # variable information
# print(data_raw.variables)

"""## Dataset Overview

We use the Breast Cancer dataset from the UCI repository. The dataset includes various
     measurements from digitized images of breast masses. Each instance is labeled as
    either benign or malignant. We begin by printing the feature names and previewing
    the first few rows of data.
"""

print("Features:", X_raw.columns.tolist())
print("Target:", y_raw.columns.tolist())

print(X_raw.head())
print(y_raw.head())

"""## Feature Selection and Preprocessing
```

```
From the complete feature set, we select five features that are typically useful in
    classification tasks in this field: radius (mean), perimeter (mean), area (mean),
    concavity (mean), and concave points (mean). These are standardized to zero mean
    and the target variable is also encoded: malignant (M) as +1 and benign (B) as -1.
"""

# Select features
features = ['radius1', 'perimeter1', 'area1', 'concavity1', 'concave_points1']
X = X_raw[features].values

# Standardize selected features
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)

# Encode target: Malignant (M) = 1, Benign (B) = -1
y = y_raw['Diagnosis'].map({'M': 1, 'B': -1}).values

"""## Linear Programming Formulation

To build a linear classifier, we formulate the task as a linear programming problem.
"""

# Add bias
X_with_bias = np.hstack([X_standardized, np.ones((X_standardized.shape[0], 1))]) # [x1
    , ..., x5, b]
n_samples, n_features_plus_bias = X_with_bias.shape
# n_samples = 100  # Limit to 100 samples for the LP
# print(n_samples)

# Construct inequality constraints for LP
# Variables: [w1, ..., w5, b,  1 , ...,  n ]
A_ub = []
b_ub = []

for i in range(n_samples):
    xi = X_with_bias[i]
    yi = y[i]
    constraint = [-yi * xij for xij in xi]
    slack = [0] * n_samples
    slack[i] = -1
    row = constraint + slack
    A_ub.append(row)
    b_ub.append(-1)

"""## Slack Variable Constraints

Each slack variable       represents how much a point violates the margin. To ensure
    they are non-negative (as required in the LP), we include additional constraints
              0. These constraints are optional in `scipy.optimize.linprog` if we set
    appropriate bounds, but we include them explicitly for clarity.
"""

# Add slack variable constraints ( _i      0)
# Note: Already handled via bounds in linprog so optional
for i in range(n_samples):
    row = [0] * (n_features_plus_bias + n_samples)
    row[n_features_plus_bias + i] = -1
    A_ub.append(row)
    b_ub.append(0)
```

```
"""## Objective Function

Our goal is to find the optimal hyperplane (defined by weights ‘w‘ and bias ‘b‘) that
    separates the two classes while minimizing the sum of slack variables ‘     ‘.
Since the decision variables include both the weights/bias and slack variables, we set
     the cost vector ‘c‘ such that only the slack variables contribute to the objective
    .
"""

# Objective function
c = [0] * n_features_plus_bias + [1] * n_samples

# Convert to numpy arrays
A_ub = np.array(A_ub)
b_ub = np.array(b_ub)
c = np.array(c)

# Display shapes to confirm setup
print("Shapes of the matrices:")
print("A_ub:", A_ub.shape)
print("b_ub:", b_ub.shape)
print("c:", c.shape)

"""## Solving the Linear Program

We use ‘scipy.optimize.linprog‘ with the ’highs’ method to solve the LP. We define
    bounds so that weights and bias are unbounded (can be negative), while slack
    variables are constrained to be     0.

After solving, we extract the optimal weights and bias, along with the slack variables
    . The objective value gives the total penalty from misclassified or margin-
    violating samples.
"""

# Define bounds for linprog: weights and bias can be negative, slack variables must be
     non-negative
bounds = [(None, None)] * n_features_plus_bias + [(0, None)] * n_samples

# Solve the LP
result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='highs')

# Extract solution
if result.success:
    w = result.x[:n_features_plus_bias - 1]
    b = result.x[n_features_plus_bias - 1]
    slack = result.x[n_features_plus_bias:]
else:
    w = b = slack = None

# Output
print("Optimal weights:", w)
print("Optimal bias:", b)
print("Objective value (sum of slack variables):", result.fun if result.success else "
    Failed")
print("Number of slack variables greater than 1:", np.sum(slack >= 1))
print("Slack variables:", slack)
```

LISTING 1. `lpsolver.py` – five-feature LP classifier

B Ten-Feature Results

The next listing (`solver2.py`) extends the feature set to the ten variables recommended in [2]. It produces the weights in Table 2 and achieves an objective value of 32 (vs 103 for the five-feature model) with only 11 slack variables ≥ 1.

Feature (standardized)	Weight w_j
texture_mean	+1.1980
area_mean	+2.9166
smoothness_mean	+0.0292
concavity_mean	+0.8081
area_se	+3.9203
concavity_se	−0.5137
fractal_dimension_se	−0.7264
smoothness_worst	+1.2154
concavity_worst	+1.3335
symmetry_worst	+0.5966
Bias b	+0.3819

Table 2. Values of weights and bias in ten-feature model.

```
import pandas as pd
import numpy as np
from scipy.optimize import linprog
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('data.csv')

# # Preview the data
# print(df.head())

# Select standardized features and target
features = [
    'texture_mean', 'area_mean', 'smoothness_mean', 'concavity_mean',
       'area_se', 'concavity_se', 'fractal_dimension_se', 'smoothness_worst',
       'concavity_worst', 'symmetry_worst'
]

X = data[features].values
# Standardize selected features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Encode target: Malignant (M) = 1, Benign (B) = -1
y = data['diagnosis'].map({'M': 1, 'B': -1}).values

# Add bias term
X_with_bias = np.hstack([X, np.ones((X.shape[0], 1))])  # [x1, ..., x5, b]
n_samples, n_features_plus_bias = X_with_bias.shape
# n_samples = 100  # Limit to 100 samples for the LP
# print(n_samples)

# Create inequality constraints for LP
# Variables: [w1, ..., w5, b,  1 , ...,  n ]
A_ub = []
b_ub = []

# Constraints: y_i * (w^T x_i + b) +  _i      1      -y_i * (w^T x_i + b) -  _i      -1
```

```
for i in range(n_samples):
    xi = X_with_bias[i]
    yi = y[i]
    constraint = [-yi * xij for xij in xi]
    slack = [0] * n_samples
    slack[i] = -1
    row = constraint + slack
    A_ub.append(row)
    b_ub.append(-1)

# Slack variable constraints:  _i      0     - _i      0
# for i in range(n_samples):
#     row = [0] * (n_features_plus_bias + n_samples)
#     row[n_features_plus_bias + i] = -1
#     A_ub.append(row)
#     b_ub.append(0)

# Objective: minimize sum of  _i
c = [0] * n_features_plus_bias + [1] * n_samples

# Convert to numpy arrays
A_ub = np.array(A_ub)
b_ub = np.array(b_ub)
c = np.array(c)

# Display shapes to confirm setup
print("Shapes of the matrices:")
print("A_ub:", A_ub.shape)
print("b_ub:", b_ub.shape)
print("c:", c.shape)

# Define bounds: weights and bias can be negative, slack variables must be non-
    negative
bounds = [(None, None)] * n_features_plus_bias + [(0, None)] * n_samples

# Compute the LP
result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='highs')

if result.success:
    # Extract the weights and bias from the result
    w = result.x[:n_features_plus_bias - 1]  # Exclude slack variables
    b = result.x[n_features_plus_bias - 1]  # Bias term
    slack = result.x[6:]  # slack variables
else:
    w = b = slack = None

print("Optimal weights:", w)
print("Optimal bias:", b)

# Print and count the number of slack variables that are greater than one
slack_greater_than_one = np.sum(slack >= 1)
print("Number of slack variables greater than one:", slack_greater_than_one)

result.success, result.message, w, b, slack

# Find the objective value (sum of slack variables)
objective_value = result.fun if result.success else None
print("Objective value (sum of slack variables):", objective_value)
```

Listing 2. `solver2.py` – ten-feature LP classifier

C Train/Test Split, PCA Visualization and Confusion Matrix

The following visualizations show a conventional train–test split (70/30) and a two-dimensional Principal-Component Analysis (five features chosen). The LP trains on the reduced space, draws decision boundaries, and reports an out-of-sample accuracy of 92.4%. The PCA scatter plot with a decision boundary and the labeled confusion matrix are included below.

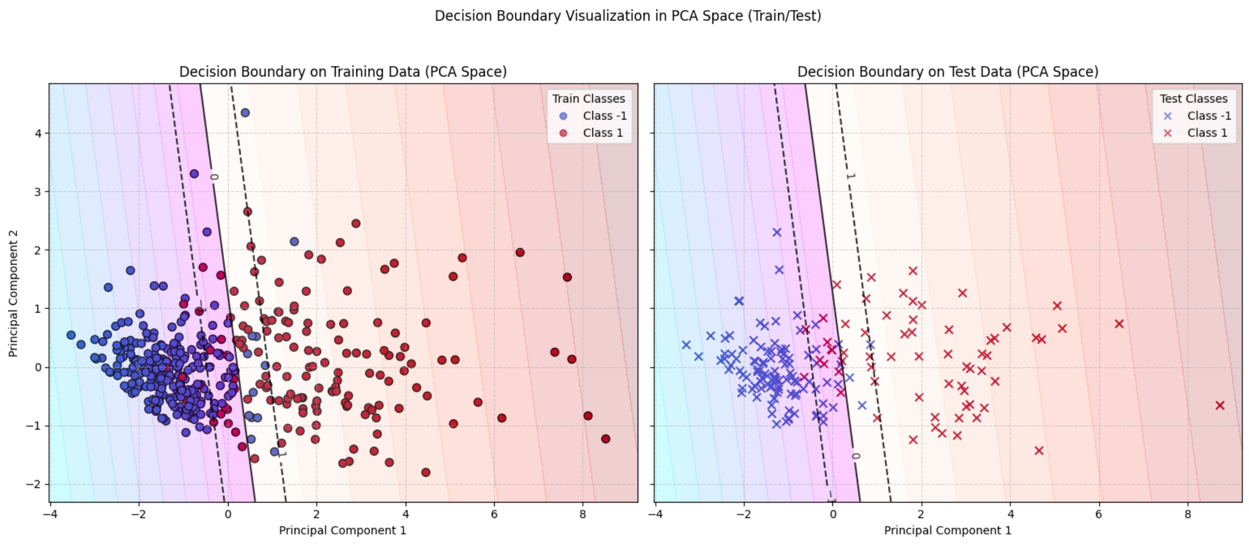

Figure 1. **Linear-programming decision surface in PCA space.** The five–feature LP classifier was trained on 70% of the WDBC data, projected onto the first two principal components.
Left: training samples with the learned decision boundary (solid black) and the ±1 margins (dashed).
Right: the same boundary applied to the 30% hold-out test set. Blue points/"×" symbols denote benign ($y = -1$), red points/"+" symbols denote malignant ($y = +1$); the colored background is the value of $f(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b$. Most points fall outside the margin band, illustrating a clear linear separation in the two-dimensional PCA subspace.

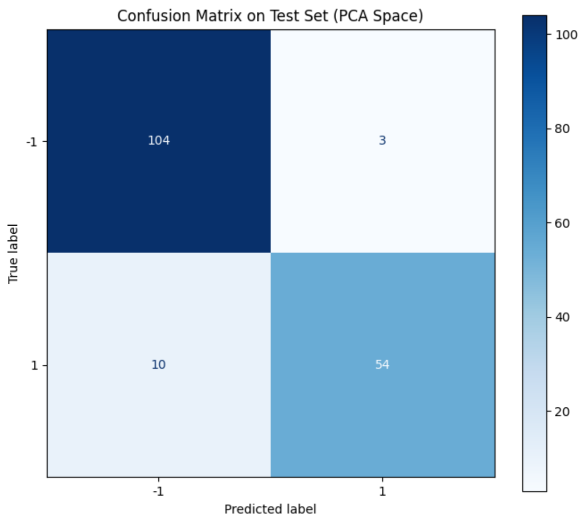

FIGURE 2. **Confusion matrix for the LP classifier on the test set (PCA space).** Out of 171 test cases, the model correctly identified 104/107 benign tumors (true negatives) and 54/64 malignant tumors (true positives), yielding a specificity of $104/(104+3) = 97.2\%$ and a sensitivity of $54/(54+10) = 84.4\%$. Misclassifications (3 false positives and 10 false negatives) cluster close to the decision boundary in Figure 1.

D Hinge Loss Linear Programming with SciPy's HiGHS Solver

In addition to our primary linear programming (LP) model, we implemented an alternative approach inspired by the hinge loss used in Support Vector Machines (SVM). This formulation penalizes both the absolute value of the weights and any misclassifications, making it well-suited for sparse and interpretable models. We solved the LP using `SciPy`'s built-in `HiGHS` solver via the `linprog` function.

Model Differences

Unlike the main model, this hinge loss LP:

- Splits each weight w_j into non-negative variables w_j^+ and w_j^-, where $w_j = w_j^+ - w_j^-$
- Introduces slack variables ξ_i to allow violations of the margin constraint
- Includes a penalty constant $C > 0$ that balances model sparsity and classification accuracy

The optimization problem is formulated as:

$$\min \sum_{j=1}^{p} (w_j^+ + w_j^-) + C \sum_{i=1}^{n} \xi_i$$

Subject to:

$$y_i \left(\sum_j (w_j^+ - w_j^-) x_{ij} + b \right) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad w_j^+, w_j^- \geq 0$$

Results

We used five features from the dataset: `radius_mean`, `perimeter_mean`, `area_mean`, `concavity_mean`, and `concave_points_mean`, all scaled to zero mean and unit variance.

The model returned the following:

- **Misclassifications:** 48 out of 569 samples
- **Error Rate:** 8.44%

- **Optimal Weights:**

$$\begin{aligned} w_1 &= 0.0000 & \text{(radius\_mean)} \\ w_2 &= 0.0000 & \text{(perimeter\_mean)} \\ w_3 &= 1.5545 & \text{(area\_mean)} \\ w_4 &= 0.4141 & \text{(concavity\_mean)} \\ w_5 &= 1.3282 & \text{(concave\_points\_mean)} \end{aligned}$$

- **Bias:** $b = -0.2006$

The model automatically discards uninformative features (weights zeroed out) and focuses on three key predictors. The negative bias term shifts the decision boundary to require stronger evidence before labeling a sample as malignant, which is suitable in medical screening to minimize false negatives.

Python Implementation (HiGHS Solver)

Listing 3. Hinge Loss LP-SVM using SciPy's HiGHS Solver

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from scipy.optimize import linprog

df = pd.read_csv("breast_cancer_data.csv")
features = ['radius_mean', 'perimeter_mean', 'area_mean',
            'concavity_mean', 'concave points_mean']
X = df[features].values
y = df['diagnosis'].map({'M': 1, 'B': -1}).values
X_scaled = StandardScaler().fit_transform(X)

n_samples, n_feats = X_scaled.shape
n_vars = 2 * n_feats + 1 + n_samples

w_pos_idx = range(n_feats)
w_neg_idx = range(n_feats, 2 * n_feats)
b_idx = 2 * n_feats
slack_idx = range(2 * n_feats + 1, n_vars)

c = np.zeros(n_vars)
C = 1.0
for j in list(w_pos_idx) + list(w_neg_idx): c[j] = 1
for i in slack_idx: c[i] = C

A_ub = []
b_ub = []

for i in range(n_samples):
    row = np.zeros(n_vars)
    xi = X_scaled[i]
    yi = y[i]
    for j in range(n_feats):
        row[w_pos_idx[j]] = -yi * xi[j]
        row[w_neg_idx[j]] = yi * xi[j]
    row[b_idx] = -yi
    row[slack_idx[i]] = -1
    A_ub.append(row)
    b_ub.append(-1)

bounds = [(0, None)] * (2 * n_feats) + [(None, None)] + [(0, None)] * n_samples

res = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='highs')
```

```python
if res.success:
    x_opt = res.x
    w = x_opt[list(w_pos_idx)] - x_opt[list(w_neg_idx)]
    b_val = x_opt[b_idx]
    scores = X_scaled @ w + b_val
    preds = np.where(scores >= 0, 1, -1)
    errors = np.sum(preds != y)
    print(f"LP-SVM misclassified {errors}/{n_samples} = {errors/n_samples:.2%}")
    print("Weights:", w)
    print("Bias:", b_val)
```

E A Non-Linear Classification Approach

Another alternative approach we explored was to minimize the sum of residual distances from a separating hyperplane. This approach not only considers whether a tumor is misclassified, but also how far it lies from the correct side of the hyperplane, introducing a more geometric view of misclassification that penalizes misclassifications more the farther they are from accuracy.

Model Differences

Unlike the main model, this residual minimizing NLP:

- Introduces residual variables r_i to measure distance from each point to the hyperplane
- Penalizes the magnitude of the misclassification rather than just the number of misclassifications
- Uses a nonlinear objective function and constraints
- Provides a geometric interpretation of misclassification severity

E.1 Formulation

This nonlinear optimization problem is formulated as:

$$\min \quad \sum_{i=1}^{N} r_i$$

subject to:

$$r_i = \frac{-y_i\left(w^T x_i + b\right)}{\|w\|}, \quad r_i \geq 0 \quad \text{for all i}$$

where:

- $x_i \in \mathbb{R}^n$ is the vector of n features from tumor i.
- $y_i \in \{-1, +1\}$ is the true label of a tumor (-1 for malignant, +1 for benign).
- $w \in \mathbb{R}^n$, $b \in \mathbb{R}$ are the normal vector and intercepts to the separating hyperplane, respectively.
- r_i is the residual distance of a misclassified point from the separating hyperplane.

The idea behind this formulation is based on geometry.

The quantity

$$\frac{\left(w^T x_i + b\right)}{\|w\|}$$

gives the distance of a point x_i from the separating hyperplane defined by $w^T x_i + b = 0$.

For a point to be classified on the correct side if the hyperplane, the sign of this distance must match the sign of the point's true label (in this case, -1 or +1).

That is, a correctly classified malignant tumor must have an associated positive distance, while a correctly classified benign tumor must have an associated negative distance.

We want residuals to be defined as positive only when a point is misclassified. This can be achieved by multiplying the point's distance from the hyperplane by the negative of the point's true classification of -1 or +1. As a result of this multiplication, correctly classified points will have negative residuals, while misclassified points will have positive ones.

This is where the constraints of this nonlinear program originate. By first defining quantitatively the value of each residual and then resitricting it to nonzero values

Limitations

This method was unfortunately never implemented due to its complexity, so we were never able to evaluate its performance as a classification model. However, it still serves as a useful way of tackling classification problems and remains a promising model for future work.