**Homework #4**

**Kavana Manvi Krishnamurthy**

**ID: 2158984**

**Pima Diabetes**

Open up HW4\_Diabetes.py

1. First, let’s run an SVM classifier.
   1. First we need to import the functions, on line 11, replace the comment with a call to import SVC () from the sklearn “SVM” package.
   2. On line 278, create a SVC(). Using the API link in the accompanying document, call that function, and pass in the following parameters:
      1. Set kernel to ‘rbf’
      2. Set gamma = ‘scale’
      3. Set C = 1.0
      4. Set probability = True
      5. Set random\_state variable to rand\_st
   3. Add in a cross\_validate function on line 279 (use previous homework as an example) with 5 folds, and pass in the clf object.

*\*Question #1a: Run the code once, record the accuracy and AUC score. What do you notice about the scores? How do they compare with previous homeworks?*

*--ML Model Output--*

*SVM Acc: 0.76 (+/- 0.04)*

*SVM AUC: 0.82 (+/- 0.07)*

*CV Runtime: 0.5315251350402832*

**Diabetes Dataset classifier results from previous homework:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Classifiers** | **Accuracy (+/-)** | **AUC (+/-)** | **CV Runtime (s)** |
| **Decision Tree (HW1)** | 0.71 (+/- 0.08) | 0.69 (+/- 0.07) | 0.04099 |
| **Random Forest (HW2)** | 0.77 (+/- 0.06) | 0.83 (+/- 0.07) | 0.8980 |
| **Gradient Boosting (HW3)** | 0.76 (+/- 0.06) | 0.82 (+/- 0.06) | 1.26 |
| **AdaBoost (HW3)** | 0.76 (+/- 0.07) | 0.83 (+/- 0.08) | 1.63 |
| **MLP (HW3)** | 0.70 (+/- 0.06) | 0.73 (+/- 0.06) | 2.65 |

***Comparison with Previous Models***

***Accuracy*** *The SVM model achieved an accuracy of 0.76, demonstrating solid performance. It outperformed both the Decision Tree (0.71) and MLP (0.70) models, showing its robustness. While it matched the accuracy of Gradient Boosting and AdaBoost (both 0.76), it was slightly below the Random Forest model (0.77), which had the highest accuracy among the classifiers tested.*

***AUC (Area Under the Curve)*** *With an AUC score of 0.82, the SVM model showed strong discriminatory power. It significantly outperformed the Decision Tree (0.69) and MLP (0.73) models, reinforcing its ability to separate classes effectively. While it matched Gradient Boosting (0.82), it was slightly behind Random Forest and AdaBoost (both 0.83) in terms of AUC.*

***Cross-Validation Range*** *The cross-validation results for SVM showed a tight accuracy range of ±0.04, indicating stable and reliable performance. Compared to other models, whose accuracy ranges varied between ±0.06 and ±0.08, the SVM model demonstrated lower variability. Its AUC range (±0.07) was comparable to other models, which mostly fell between ±0.06 and ±0.08, confirming its consistency.*

***Key Takeaways*** *The SVM model performed competitively, standing on par with ensemble methods like Gradient Boosting and AdaBoost. It showed clear improvements over simpler models such as Decision Tree and MLP, proving its effectiveness in classification. Additionally, the tight accuracy range (±0.04) suggests that SVM maintains consistent performance across different validation folds.*

***Efficiency*** *The runtime of 0.53 seconds highlights the efficiency of the SVM model. While it is faster than most other models tested, the Decision Tree model was the only one that ran more quickly. Given its balance of speed and accuracy, SVM emerges as a strong and reliable classifier among the tested models.*

*\*Question #1b: In the Scikit API for SVC, it explains the probability parameter … why did we set it equal to ‘True’? What does that do?*

SVC and NuSVC’s decision\_function spits out per-class scores for each sample (or just one score in binary classification). If you turn on probability=True, you get class probabilities via predict\_proba, but for binary cases, it uses **Platt scaling** (logistic regression on SVM scores with extra cross-validation). In multi-class, it extends beyond that. **Heads up**: Platt scaling can be slow on big datasets and sometimes the probability estimates don’t match the raw scores (e.g., argmax of scores ≠ argmax of probabilities). Also, predict might label a sample as positive even if predict\_proba is < 0.5, and vice versa. If you just need confidence scores (not necessarily probabilities), skip the extra overhead—set probability=False and use decision\_function instead. Oh, and if you're using decision\_function\_shape='ovr' with multiple classes, predict doesn’t break ties unless you set break\_ties=True, but that adds some extra computation.

1. Let’s explore how changing some of the parameters for the SVM
   1. Set gamma = ‘auto’
   2. Change the kernel parameter of the SVC() to ‘sigmoid’
   3. Now set the kernel to ‘linear’

*\*Question #2: Run the code once for each setting of the kernel, record the accuracy and AUC scores. What do you notice about the scores compared to Question #1? What about run-times?*

Answer:

|  |  |  |  |
| --- | --- | --- | --- |
| Kernel | Accuracy (±) | AUC (±) | CV Runtime (s) |
| Sigmoid | 0.65 (±0.00) | 0.50 (±0.00) | 0.25 |
| RBF | *0.76 (+/- 0.04)* | *0.82 (+/- 0.07)* | 0.531 |
| **Linear** | 0.77 (±0.05) | 0.83 (±0.05) | 82.05 |

1. Accuracy:
   * Linear kernel performs slightly better (0.77) than RBF (0.76)
   * Sigmoid kernel performs significantly worse (0.65)
2. AUC:
   * Linear kernel shows a slight improvement (0.83) over RBF (0.82)
   * Sigmoid kernel performs poorly with an AUC of 0.50 (equivalent to random guessing)
3. Cross-validation ranges:
   * RBF: Accuracy ±0.04, AUC ±0.07
   * Sigmoid: No variation (±0.00 for both)
   * Linear: Accuracy ±0.05, AUC ±0.05

**Runtime Comparison**

1. Sigmoid kernel is the fastest (0.25 seconds)
2. RBF kernel is moderately fast (0.53 seconds)
3. Linear kernel is significantly slower (82.05 seconds)

The Linear kernel edges out RBF in accuracy and AUC, but at a massive runtime cost—way slower than both RBF and Sigmoid, likely due to how it scales with the dataset. Sigmoid? A total flop. It performs poorly and shows zero variation in cross-validation, meaning it's probably not fitting well at all. Meanwhile, RBF holds its own, delivering solid performance with a reasonable runtime, making it the best balance for real-world use. If time isn’t an issue, Linear might be worth it, but for efficiency, RBF is the way to go.

1. Finally, let’s run feature selection again on the Diabetes dataset, but this time do it using SVMs. Since SVMs depending on the kernel type can function like a linear regression method, and produce coefficients that we can use as a measure of “feature importance” natively.
   1. Let’s leave the SVC() kernel set as ‘linear’
   2. To turn on feature selection, we need to first on line 38 change the feat\_select flag to equal 1 instead of 0
   3. Note that there is an option to change the feature selection type is already set to 2 (wrapper-based) on line 39
   4. You will need to add a SVC(), call to pass to the clf object on line 191, you can use something similar to the calls used elsewhere in the code. Don’t forget to set the parameters, particularly the kernel.
   5. Make sure you set the kernel to the ‘linear’ when using it for feature selection (so it produces coefficients for FS), or your code will give errors.

*\*Question #3a: Run the code once, record the accuracy and AUC scores. What do you notice about the scores? How do they compare to the performance Question 2 above for SVMs using a linear kernel with no feature selection?*

*Answer:*

SVM Acc: 0.65 (+/- 0.00)

SVM AUC: 0.50 (+/- 0.25)

CV Runtime: 0.3916170597076416

|  |  |  |  |
| --- | --- | --- | --- |
| Feature selection(Linear kernel) | Accuracy (±) | AUC (±) | CV Runtime (s) |
| No | 0.77 (±0.05) | 0.83 (±0.05) | 82.05 |
| Yes | 0.65 (+/- 0.00) | 0.50 (+/- 0.25) | 0.3916170597076416 |

1. Accuracy:
   * Feature selection led to a significant drop in accuracy from 0.77 to 0.65
   * The accuracy with feature selection is notably lower
2. AUC:
   * Feature selection resulted in a dramatic decrease in AUC from 0.83 to 0.50
   * An AUC of 0.50 suggests the model is performing no better than random guessing
3. Cross-validation ranges:
   * Without feature selection: Accuracy ±0.05, AUC ±0.05
   * With feature selection: Accuracy ±0.00, AUC ±0.25
4. Runtime: significantly lower when feature selection is turned on.

The feature selection process significantly degraded the model’s performance, with both accuracy and AUC dropping substantially. The accuracy showing no variation (±0.00) is concerning, suggesting potential issues with model fitting, while the widened AUC range (±0.25 vs. ±0.05) indicates high variability across folds. Although feature selection drastically reduced runtime from 82.05s to 0.39s, the resulting model is ineffective, with an AUC of 0.50, implying near-random predictions. The lack of accuracy variation further suggests the model may be predicting a single class consistently. This outcome highlights the risk of removing critical predictive features, emphasizing the need for careful evaluation of feature selection techniques to maintain model integrity and performance.

*\*Question #3b: What features were selected, and which were removed? Were there any differences from when you did feature selection with Boosting in HW3, or Random forests in HW2?*

*Answer:*

***HW4 SVM:***

*Selected [‘Family History’]*

*Features not selected: [‘’Times Pregnant’, ‘Blood Glucose’, ‘Blood Pressure’, ‘Skin Fold Thickness’, ‘2-Hour Insulin’, ‘BMI’, ‘Family History’, ‘Age’]*

***Boosting HW3:***

Selected ['Blood Glucose', 'BMI', 'Age']

Features (total/selected): 8 3

Features not selected: ['Times Pregnant', 'Blood Pressure', 'Skin Fold Thickness', '2-Hour Insulin', 'Family History']

***Random Forest HW2:***

Selected features: 'Blood Glucose', 'BMI', 'Family History', 'Age'  
Removed features: 'Class', 'Times Pregnant', 'Blood Pressure', 'Skin Fold Thickness', '2-Hour Insulin'

Features (total/selected): 8 4

** ***Consistent Key Features:*** *'Blood Glucose', 'BMI', and 'Age' were important in both Boosting and Random Forest models, indicating their strong predictive power.*

** ***Feature Selection Differences:*** *SVM selected only 'Family History', while Boosting focused on 'Blood Glucose', 'BMI', and 'Age'. Random Forest included all these plus 'Family History'.*

** ***Commonly Excluded Features:*** *'Times Pregnant', 'Blood Pressure', 'Skin Fold Thickness', and '2-Hour Insulin' were consistently not selected, suggesting they had minimal impact on predictions.*

** ***Model-Specific Trends:*** *SVM's reliance on 'Family History' suggests it found this feature particularly relevant in a linear space, whereas Boosting and Random Forest emphasized metabolic and demographic factors.*

*Different models interpret feature importance in unique ways. SVM, using a linear kernel, zeroed in on just* ***one feature*** *('Family History'), likely because it offered the best linear separation. Meanwhile,* ***Boosting (3 features)*** *and* ***Random Forest (4 features)*** *captured more complex, non-linear relationships, emphasizing metabolic and demographic factors like 'Blood Glucose', 'BMI', and 'Age'. Since ensemble methods can leverage multiple features without overfitting, they tend to pick up more patterns in the data. While there's clear overlap—especially between Boosting and Random Forest—each model brings its own perspective, proving why testing multiple approaches is key to understanding feature importance.*

**Wine Quality Dataset**

Open up HW4\_Wine.py … First, let’s repeat the steps we did above for Diabetes.

1. First, let’s run an SVM regressor.
   1. First we need to import the functions, on line 12, add a call to import SVR() from the sklearn “SVM” package.
   2. On line 368, create a SVR(). Using the API link in the accompanying document, call that function, and pass in the following parameters:
      1. Set kernel to ‘rbf’
      2. Set gamma = 0.1
      3. Set C = 1.0
   3. Add in a cross\_validate function on line 369 (use previous homework as an example) with 5 folds, and pass in the rgr object.

*\*Question #4a: Run the code once, record the RMSE and Explained Variance.*

*--ML Model Output–*

*SVM RMSE:: 0.77 (+/- 0.06)*

*SVM Expl Var: 0.08 (+/- 0.10)*

*CV Runtime: 1.0557501316070557*

*\*Question #4b: In the Scikit API for SVR, you will notice there is no probability parameter (averse to for the classifier version), why do you think that is?*

SVR in Scikit-learn doesn’t have a probability parameter because regression and classification fundamentally differ. Unlike SVC, which predicts discrete class labels with associated probabilities, SVR outputs continuous values where probability doesn’t have a direct interpretation. Regression models focus on point estimates or prediction intervals rather than class likelihoods, making probability estimation unnecessary. SVR’s objective is to minimize deviations within a margin, and methods like Platt scaling (used in SVC for probability calibration) aren’t applicable. Additionally, probability estimation introduces computational overhead, which isn’t meaningful for regression tasks. Keeping SVR probability-free ensures a clear distinction between regression and classification while optimizing performance.

1. Let’s explore how changing some of the parameters for the SVM
   1. Change the kernel parameter of the SVR() to ‘sigmoid’
   2. Now set the kernel to ‘linear’

*\*Question #5: Run the code once for each setting of the kernel, record the RMSE and Explained Variance. What do you notice about the scores compared to Question #4 RBF? What about run-times?*

|  |  |  |  |
| --- | --- | --- | --- |
| Kernel | SVM RMSE (±) | SVM Expl Var (±) | CV Runtime (s) |
| Sigmoid | 0.91 (±0.22) | 0.00 (±0.00) | 0.53 |
| Linear | 0.66 (±0.02) | 0.29 (±0.19) | 21.65 |
| RBF | 0.77 (±0.06) | 0.08 (±0.10) | 1.06 |

***Performance:*** *The linear kernel delivers the best results, achieving the lowest RMSE (0.66) and highest explained variance (0.29), making it the strongest performer. The RBF kernel follows closely behind with a slightly higher RMSE (0.77), while the sigmoid kernel performs the worst, with the highest RMSE (0.91) and no explained variance, indicating poor predictive power.*

***Consistency:*** *The linear kernel is the most stable, showing minimal RMSE variation (±0.02) but a wider range in explained variance (±0.19). The RBF kernel demonstrates moderate consistency across both metrics. In contrast, the sigmoid kernel is highly inconsistent, with significant RMSE variation (±0.22) and no change in explained variance, suggesting it struggles with generalization.*

***Runtime:*** *The sigmoid kernel is the fastest, taking just 0.53 seconds, making it the most computationally efficient. The RBF kernel is moderately fast at 1.06 seconds, balancing speed and performance. However, the linear kernel is significantly slower, requiring 21.65 seconds, which may be a concern for scalability despite its strong predictive performance.*

***Comparison to RBF:*** *The linear kernel outperforms RBF in both RMSE and explained variance, making it the superior choice when accuracy is the priority. The sigmoid kernel falls behind RBF across all metrics, making it the weakest option. Meanwhile, RBF strikes a balance, offering decent accuracy with much faster runtime than the linear kernel, making it a practical middle ground.*

*While the linear kernel shows the best performance metrics, the choice between linear and RBF kernels would depend on the specific needs of the project, balancing performance gains against computational costs. The sigmoid kernel, despite its speed, is not recommended due to its poor performance on this dataset.*

1. Discriminant Analysis methods can be prone to issues with data distributions, so let’s see if normalizing the features has any effect.
   1. Let’s leave the SVC() kernel set as ‘linear’
   2. On line 35, change the norm\_features flag to equal 1 instead of 0

*Question #6: Run the code once, record the RMSE and Explained Variance. What do you notice about the scores, or the run times? How do they compare to results in Question #5?*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Preprocessing & Kernel | SVM RMSE | Std Dev (RMSE) | SVM Expl Var | Std Dev (Expl Var) | CV Runtime |
| No Normalization + Linear | 0.66 | ±0.02 | 0.29 | ±0.19 | 21.65 |
| Normalization + Linear | 0.66 | ±0.02 | 0.29 | ±0.16 | 1.1861 |

1. Performance Metrics:
   * RMSE is identical (0.66) for both normalized and non-normalized data
   * Explained Variance is also the same (0.29) in both cases
2. Consistency (Standard Deviations):
   * RMSE standard deviation is identical (±0.02) for both
   * Explained Variance shows slightly less variation with normalization (±0.16 vs ±0.19)
3. Runtime:
   * Normalization dramatically reduced the runtime from 21.65 seconds to 1.1861 seconds
   * This represents an approximately 18x speedup

Normalization didn’t change the RMSE or Explained Variance, showing that the linear SVR handles feature scaling like a champ. But it did make the Explained Variance more consistent (smaller standard deviation), meaning a little extra stability. The biggest win? A massive speed boost—normalization slashed runtime without messing with performance. That’s a game-changer for scaling up to bigger datasets or when you just need faster training and predictions. Bottom line: SVR sees the same patterns whether you normalize or not, but if you want speed and efficiency, normalization is the move.

1. Let’s run feature selection again on the Wine dataset, just like we did for Diabetes above.
   1. Let’s leave the norm\_features flag turned on (set to 1)
   2. To turn on feature selection, we need to first on line 38 change the feat\_select flag to equal 1 instead of 0
   3. Note that there is an option to change the feature selection type is already set to 2 (wrapper-based) on line 39
   4. You will need to add a SVR(), call to pass to the rgr object on line 254, you can use something similar to the calls used elsewhere in the code. Don’t forget to set the parameters, particularly the kernel.
   5. Make sure you set the kernel to the ‘linear’ when using it for feature selection (so it produces coefficients for FS), or your code will give errors.

*\*Question #7a: Run the code once Record the RMSE and Explained Variance Score. What do you notice about the scores? How do they compare to performance in Question 6 above for SVMs using a linear kernel with no feature selection?*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Preprocessing & Kernel | SVM RMSE | Std Dev (RMSE) | SVM Expl Var | Std Dev (Expl Var) | CV Runtime |
| No Feature Selection + Linear | 0.66 | ±0.02 | 0.29 | ±0.16 | 1.1861 |
| Feature Selection + Linear | 0.68 | ±0.04 | 0.27 | ±0.25 | 0.6306 |

1. RMSE:
   * Slightly increased from 0.66 to 0.68 with feature selection
   * Standard deviation doubled from ±0.02 to ±0.04
2. Explained Variance:
   * Decreased slightly from 0.29 to 0.27 with feature selection
   * Standard deviation increased significantly from ±0.16 to ±0.25
3. Runtime:
   * Reduced by about 47% from 1.1861 to 0.6306 seconds with feature selection
4. Overall Performance:
   * Feature selection led to a slight decrease in model performance
   * Increased variability in both RMSE and Explained Variance scores

Feature selection made the model a bit worse—higher RMSE, lower Explained Variance, and more variability in predictions. But the real win? Speed. It nearly cut the runtime in half, making things way more efficient. So, there’s a trade-off: you lose a little accuracy and consistency, but you gain serious speed. The fact that performance didn’t drop too much means the key features were kept, but some useful info probably got left behind. If you’re all about efficiency and can handle a slight performance dip, feature selection might be worth it.

*Question #7b: What features were selected, and which were removed?* *How did this compare with features selected in previous homeworks (Random Forests, Gradient Boosting)?*

**HW4 SVM:**

Selected : [‘volatile acidity’, ‘sulphates’, ‘alcohol’]

Features (total/selected): 11 3

Features not Selected : [‘Class’, ‘fixed acidity’, ‘citric acid’, ‘residual sugar’, ‘chlorides’, ‘free sulfur dioxide’, ‘total sulfur dioxide’, ‘density’, ‘pH’]

**Hw2 Random forest:**

*Removed features:*  *'fixed acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH'*

Selected features: *'volatile acidity', 'sulphates', 'alcohol'*

**HW3 Boosting:Binnig=0**

Selected ['volatile acidity', 'sulphates', 'alcohol']

Features (total/selected): 11 / 3

Features not selected: [ 'fixed acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH']

**Consistency Across Models:**

All three models—SVM, Random Forest, and Boosting—picked the exact same features: 'volatile acidity', 'sulphates', and 'alcohol'. This level of consistency across different algorithms is a solid indicator that these three features are the top dogs when it comes to predicting wine quality in this dataset. When multiple models agree on the most important features, it really highlights their relevance.

**Features Consistently Removed:**

Every model was on the same page when it came to dropping certain features: 'fixed acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', and 'pH'. This unanimous removal suggests these variables just don’t hold much value in predicting wine quality here. They might not have enough power to make a meaningful impact on the model, so it’s a clear signal to leave them out.

In conclusion, the feature selection results show remarkable consistency across SVM, Random Forest, and Boosting methods for the Wine dataset. All three models identified 'volatile acidity', 'sulphates', and 'alcohol' as the most important predictors of wine quality, while consistently removing the other features.

1. Let’s compare some other feature selection methods, such as using mutual information between each feature and the target (based on information theory and entropy).
   1. Set fs\_type line 39 to 3

*\*Question #8a: Run the code once Record the RMSE and Explained Variance Score. What do you notice about the scores? How do they compare to performance above using wrapper feature selection in Question 7?*

Univariate Feature Selection – Mutual Info: Ranked Features

0 alcohol : 0.21444185980042985

1 sulphates : 0.13225606262942513

2 density : 0.10442485307285665

3 volatile acidity : 0.09732951365760911

4 total sulfur dioxide : 0.06985834205706443

5 citric acid : 0.06469832802339415

6 fixed acidity : 0.06046914182081675

7 pH : 0.04733945766192971

8 chlorides : 0.04441637207318827

9 residual sugar : 0.041993167315267144

10 free sulfur dioxide : 0.005973330384507314

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Preprocessing & Kernel | SVM RMSE | Std Dev (RMSE) | SVM Expl Var | Std Dev (Expl Var) | CV Runtime |
| No Feature Selection | 0.66 | ±0.02 | 0.29 | ±0.16 | 1.1861 |
| Feature Selection 2 – simple wrapper | 0.68 | ±0.04 | 0.27 | ±0.25 | 0.6306 |
| Feature Selection Type 3- univariate mutual info | 0.68 | ±0.04 | 0.26 | ±0.25 | 0.6593 |

* *The RMSE of 0.68 indicates the average prediction error in the same units as the target variable. The relatively small standard deviation (+/- 0.04) suggests consistency across cross-validation folds.*
* *The score of 0.26 means that the model explains approximately 26% of the variance in the target variable. This is a relatively low score, indicating that the model’s predictions don’t account for much of the variability in the data.The high standard deviation (+/- 0.25) suggests inconsistency across folds, which could indicate overfitting or high sensitivity to the specific data in each fold.*
* *The cross-validation process took about 0.66 seconds, which is relatively quick*

*Feature Selection Type 3 – simple wrapper method performs almost the same as the previous feature selection method—same RMSE, with just a slight dip in Explained Variance. There’s no real improvement here; if anything, it’s a tiny step back. Both feature selection approaches seem to impact error rates the same way, meaning the choice of method doesn’t drastically change performance. On the bright side, Type 3 keeps the runtime efficiency gains, staying way faster than running the model with all features. But the high variance in Explained Variance (±0.25) is still a red flag, possibly hinting at overfitting or sensitivity to different data splits. Bottom line? Type 3 doesn’t really bring anything new to the table—it’s just as efficient, but not any better at making predictions.*

*Question #8b: What features were selected, and which were removed?*

Selected: [‘volatile acidity’, ‘chlorides’, ‘density’, ‘sulphates’, ‘alcohol’]

Features (total/selected): 11 5

Removed: [‘fixed acidity’, ‘citric acid’, ‘residual sugar’, ‘free sulfur dioxide’, ‘total sulfur dioxide’, ‘pH’]

1. Let’s run feature selection again on the Wine dataset, except using a full-blown wrapper. I’ve already written the helper function for you (line 55), you just need to turn it on. This is a very straightforward exhaustive search method, with no regularization, so it will probably pick more features than truly necessary. Warning: it can take a while to run, perhaps a couple minutes depending on your computer.
   1. Set fs\_type line 39 to 4
   2. You will need to add a SVR(), call to pass to the rgr object on line 304, you can use something similar to the calls used elsewhere in the code. Don’t forget to set the parameters, particularly the kernel.

*\*Question #9a: Run the code once Record the RMSE and Explained Variance Score. What do you notice about the scores? How do they compare to performance above for feature selection using the simple wrapper in Question 7 and the univariate mutual info in Question 8?*

*Full blown Wrapper Feat Sel Runtime: 188.72298526763916*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Preprocessing & Kernel | SVM RMSE | Std Dev (RMSE) | SVM Expl Var | Std Dev (Expl Var) | CV Runtime |
| No Feature Selection | 0.66 | ±0.02 | 0.29 | ±0.16 | 1.1861 |
| Feature Selection 2 – Simple Wrapper | 0.68 | ±0.04 | 0.27 | ±0.25 | 0.6306 |
| Feature Selection Type 3 – Univariate Mutual Info | 0.68 | ±0.04 | 0.26 | ±0.25 | 0.6593 |
| Feature Selection Method 4 – Full Blown Wrapper | 0.66 | ±0.02 | 0.30 | ±0.18 | 1.8403 |

*Question #9b: What features were selected, and which were removed?*

*Selected [‘volatile acidity’, ‘residual sugar’, ‘chlorides’, ‘free sulfur dioxide’, ‘total sulfur dioxide’, ‘density’, ‘pH’, ‘sulphates’, ‘alcohol’]*

*Features (total/selected): 11 9*

1. You might note the long run times for Question #9. You can imagine that would be even more exacerbated with different kernels or ML methods. Given that context, look at the comment on line 56. No changes are needed for this question, norcode to run here.

*\*Question #10a:* *Based on the comment on line 56, explain how we might search the feature set space in a more optimal manner.*

*Line 56 suggests the use of greedy search, random search, genetic algorithms etc.*

*Let’s first talk about Greedy search. It is great, we either do backward elimination or forward selection but might get stuck in a local optimum. There is no reason to add/remove features if performance is not increasing significantly. That is where Regularization comes in- it decreases the complexity and prevents overfitting.*

***How regularization helps****: Regularization reduces complexity by penalizing large coefficients, ensuring the model does not overfit even with suboptimal feature subsets.*

*Random search incorporates searching for random subsets of features, but it requires significant computational effort. It’s essentially like winning the lottery—finding the best subset by chance.****How regularization helps****: Regularization reduces complexity by discouraging reliance on irrelevant or redundant features, even when random subsets are explored.*

*Genetic algorithms are a more systematic way to explore the feature subset space. They simulate natural selection by evolving populations of feature subsets through crossover, mutation, and selection based on a fitness function (e.g., model accuracy or RMSE). However, they are computationally intensive due to repeated model evaluations.****How regularization helps****: Regularization reduces complexity by shrinking less important feature contributions, allowing the algorithm to focus on meaningful subsets without overfitting.*

*In all cases, regularization acts as a safeguard against overfitting and ensures that the resulting models remain interpretable and generalizable, regardless of the feature selection method used.*

*\*Question #10b: If you uncomment the print statements on lines 87 and 96, and watch the code run, you may notice that there are actually several different feature sets that perform nearly the same as the optimal feature set, some of which have much fewer features than others. What is one way we could force the wrapper method to select smaller feature sets, even if they have slightly less performance? (HINT: line 56 also mentions something about this)*

Regularization is crucial regardless of the feature selection algorithm used, as it helps prevent overfitting and promotes simpler models. By incorporating regularization, we can encourage the wrapper method to select smaller feature sets, even if they have slightly lower performance. This approach is particularly beneficial for several reasons:

1. Dataset characteristics and feature selection methods:

* Large datasets with many features: Filter methods are often preferred due to their computational efficiency. However, they may miss feature interactions.
* Small to medium-sized datasets: Wrapper methods can be more effective as they consider feature interactions, but they risk overfitting.
* Datasets with complex feature relationships: Embedded methods, like LASSO, can be ideal as they combine feature selection with model training.

1. Regularization and smaller feature subsets:

Regularization inherently promotes the selection of smaller feature subsets by:

* Penalizing large coefficients: This discourages the model from relying too heavily on any single feature.
* Shrinking less important features to zero: Especially with L1 regularization (LASSO), which can effectively perform feature selection.
* Balancing bias and variance: By reducing model complexity, regularization helps find a sweet spot between underfitting and overfitting.

1. Preventing overfitting:

Regularization prevents overfitting by:

* Reducing model complexity: This improves generalization to unseen data.
* Encouraging feature sparsity: Leading to more interpretable models.
* Stabilizing coefficient estimates: Particularly useful when dealing with multicollinearity.

By incorporating regularization into the feature selection process, we can guide the wrapper method towards selecting smaller, more robust feature sets. This approach ensures that the selected features are not only predictive but also contribute to a model that generalizes well to new data.

**Summary Questions**

*\*Question #11: Line up the results from Homeworks 1,2,3,4 as a table for both Diabetes and Wine, with rows of performance metrics for each ML method (Decision Trees, Random Forests, Gradient Boosting, Ada Boost, Neural Networks, SVMs). Looking at this table of performance metrics, how would you explain the table to a boss or customer.*

**Diabetes Dataset:**

*We're looking at two key metrics: Accuracy (how often the model is correct) and AUC (Area Under the Curve, which measures the model's ability to distinguish between classes). Both metrics range from 0 to 1, with higher values indicating better performance. The (+/-) values show the variation in performance across different data splits.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Classifiers** | **Accuracy (+/-)** | **AUC (+/-)** | **CV Runtime (s)** |
| **Decision Tree (HW1)** | 0.71 (+/- 0.08) | 0.69 (+/- 0.07) | 0.04099 |
| **Random Forest (HW2)** | 0.77 (+/- 0.06) | 0.83 (+/- 0.07) | 0.8980 |
| **Gradient Boosting (HW3)** | 0.76 (+/- 0.06) | 0.82 (+/- 0.06) | 1.26 |
| **AdaBoost (HW3)** | 0.76 (+/- 0.07) | 0.83 (+/- 0.08) | 1.63 |
| **MLP (HW3)** | 0.70 (+/- 0.06) | 0.73 (+/- 0.06) | 2.65 |
| SVM RBF Kernel(HW4) | *0.76 (+/- 0.04)* | *0.82 (+/- 0.07)* | 0.531 |

For the Diabetes dataset, Random Forest, Gradient Boosting, AdaBoost, and SVM all performed similarly well, achieving accuracies of around 76-77% and AUC values of ~82-83%.

**Dataset characteristics:** Dataset For smaller datasets like Diabities, ensemble tree-based methods tend to perform better than Neural Networks, which typically require larger datasets to reach their full potential. SVM also performed exceptionally well on both datasets due to its ability to handle high-dimensional spaces effectively.

 **Performance &** **Computational Trade-offs:**

* Decision Trees: These are like simple flowcharts that make quick decisions. They're the fastest to run (0.04 seconds) but less accurate (71%), making them fast but less accurate compared to ensemble methods.
* Random Forest: This method uses many decision trees together, like a forest of decision-makers. It's more accurate (77%) and still reasonably fast (0.9 seconds), offering a good balance between computational speed and accuracy.
* Gradient Boosting and AdaBoost: These are advanced methods that learn from their mistakes to improve accuracy. They're slightly slower (1.26 and 1.63 seconds) but very accurate (76%), making them great choices when you need high-quality predictions and have a bit more time. Boosting methods like Gradient Boosting and AdaBoost are slightly slower but deliver comparable performance.
* Neural Networks (MLP): This mimics how the human brain works. It's the slowest (2.65 seconds) and less accurate (70%) here, likely because it needs more data to perform well. It's powerful but requires more fine-tuning and data to shine.
* SVM: This method finds the best way to create hyperplanes that separate different groups in the data. It's both fast (0.53 seconds) and accurate (76%), making it a strong all-rounder that balances speed and performance well.

**Ensemble Methods Lead the Pack:** Random Forest, Gradient Boosting, and AdaBoost performed the best in terms of accuracy and AUC, showing the strength of ensemble learning in improving model reliability. These models consistently outperformed individual Decision Trees and Neural Networks due to their ability to combine multiple weak learners into a stronger model. These methods are particularly effective for capturing complex relationships in the data while reducing overfitting.

**Balancing Bias and Variance:** Bagging (Random Forest) and boosting methods (Gradient Boosting, AdaBoost) were equally effective, indicating that reducing variance (RF) and reducing bias (Boosting) both helped in this dataset.

In conclusion, for this Diabetes classification task, ensemble methods (Random Forest, Gradient Boosting, AdaBoost) and SVM with RBF Kernel offer the best balance of accuracy and AUC.

**Wine Dataset:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **RMSE (± Std)** | **Explained Variance (± Std)** | **CV Runtime (seconds)** | **Accuracy (± Std)** | **AUC (± Std)** |
| **Decision Tree Regressor (HW1)** | 0.90 (± 0.10) | -0.31 (± 0.17) | 0.0684 | - | - |
| **Random Forest Regressor (HW2) at 50 Trees** | 0.65 (± 0.03) | 0.32 (± 0.11) | 1.103 | - | - |
| **Gradient Boosting Regressor** | 0.64 (± 0.01) | 0.34 (± 0.12) | 1.43 | - | - |
| **AdaBoost Regressor** | 0.66 (± 0.03) | 0.31 (± 0.16) | 2.01 | - | - |
| **Neural Network Regressor** | 0.66 (± 0.05) | 0.30 (± 0.08) | 3.96 | - | - |
| **Gradient Boosting Classifier** | - | - | 2.39 | 0.73 (± 0.05) | 0.81 (± 0.06) |
| **AdaBoost Classifier** | - | - | 1.83 | 0.73 (± 0.05) | 0.82 (± 0.07) |
| **Multi-Layer Perceptron Classifier** | - | - | 6.22 | 0.73 (± 0.06) | 0.81 (± 0.06) |
| ****SVM Regressor-RBF kernel**** | 0.77 (±0.06) | 0.08 (±0.10) | 1.06 |  |  |

*In a regression task, we're trying to predict a continuous value, like predicting wine quality on a scale from 1 to 10, rather than classifying something into categories.* ***RMSE (Root Mean Square Error)*** *shows how far off our predictions are from the true values, with a lower number meaning better accuracy in predictions.* ***Explained Variance*** *measures how much of the variation in the data can be explained by the model, with higher values being better.* ***CV Runtime*** *is the time it takes for the model to be trained and tested, giving us an idea of how efficient the model is. For classifiers,* ***Accuracy*** *tells us the percentage of correct predictions, while* ***AUC (Area Under the Curve)*** *measures the model's ability to distinguish between different classes, with a higher AUC being better. Let’s now dive into each model to see how they perform.*

***Decision Tree Regressor:***

*The Decision Tree**Regressor works by making predictions based on a series of yes/no questions, creating a tree structure where each decision point splits the data into different groups. It is the fastest model in this set, taking just 0.07 seconds to run, but it comes with the trade-off of being the least accurate, with an RMSE of 0.90. While it provides quick results, it is not reliable for precise predictions, making it suitable for situations where speed is more important than accuracy.*

***Random Forest Regressor:***

*The* ***Random Forest Regressor*** *improves upon the Decision Tree by combining many decision trees to produce a more robust and reliable prediction. It strikes a good balance between accuracy and computational efficiency, with an RMSE of 0.65 and a runtime of 1.1 seconds. This model is great for many applications, as it balances speed with strong performance, making it a go-to for tasks requiring dependable predictions without excessive computational cost.*

***Gradient Boosting Regressor:***

*The* ***Gradient Boosting Regressor*** *builds trees sequentially, where each tree corrects the mistakes of the previous one. This approach results in the best accuracy for regression in this set, with an RMSE of 0.64, though it does come with a slightly slower runtime of 1.43 seconds. While it delivers excellent predictive power, it is more complex and slower than* ***Random Forest****, making it ideal for applications where top-tier accuracy is essential, and some additional computational time can be afforded.*

***AdaBoost Regressor:***

*The* ***AdaBoost Regressor*** *focuses on the harder-to-predict cases by adjusting its predictions based on misclassified examples. It offers good accuracy with an RMSE of 0.66, though its runtime is the slowest of the tree-based methods at 2.01 seconds. It’s particularly good at handling tough predictions and improving overall performance, but its longer runtime may make it less ideal for applications where speed is critical.*

***Neural Network Regressor:***

*The* ***Neural Network Regressor*** *mimics brain function by learning complex patterns within the data. While it can handle very complex data, it is the slowest model in this group, taking 3.96 seconds to run, and its accuracy is similar to other models, with an RMSE of 0.66. Neural Networks are powerful but require more data and fine-tuning to perform well. They are ideal for complex tasks but come with the downside of requiring significant computational time and resources.*

***Gradient Boosting Classifier:***

*The* ***Gradient Boosting Classifier*** *applies the same sequential tree-building technique used in regression but is optimized for classification tasks, in this case, categorizing wine qualities. It delivers good accuracy (73%) and an AUC of 0.81, with a moderate runtime of 2.39 seconds. This model strikes a solid balance between speed and performance, making it effective for classification tasks where strong predictive power is needed without extreme computational demands.*

***AdaBoost Classifier:***

*The* ***AdaBoost Classifier*** *is similar to the Gradient Boosting Classifier in its approach, focusing on misclassified examples to improve performance. It achieves similar classification accuracy (73%) and AUC (0.82) but has a faster runtime of 1.83 seconds. This makes* ***AdaBoost*** *an excellent choice for applications where quick and accurate classification is needed, offering a good compromise between speed and prediction quality.*

***Multi-Layer Perceptron Classifier:***

*The* ***Multi-Layer Perceptron (MLP) Classifier*** *is a more complex neural network used for classifying wine qualities. It performs similarly to the other classifiers in terms of accuracy (73%) and AUC (0.81) but is the slowest to run, taking 6.22 seconds. While powerful for identifying complex patterns in data, it requires significantly more time and tuning to reach optimal performance, making it best suited for tasks where performance justifies the extra computational cost.*

***SVM Regressor (RBF Kernel):***

*The* ***SVM Regressor*** *with an RBF kernel finds the best way to separate wine qualities in a complex, high-dimensional space. It achieves a moderate accuracy with an RMSE of 0.77 and has a good runtime of 1.06 seconds. This makes* ***SVM*** *a strong all-rounder, balancing speed and performance well, making it ideal when you need a relatively fast model that still delivers reasonable accuracy without requiring complex tuning.*

***Conclusion:***

In conclusion, for this Wine dataset, the models that excel in regression tasks, such as **Random Forest**, **Gradient Boosting**, and **AdaBoost**, provide solid accuracy and good predictive power, balancing performance and computational efficiency. **Neural Networks** and **SVM** show more complex behavior, with neural networks requiring more data and time to perform well, while **SVM** offers a good balance between speed and moderate accuracy. For classification tasks, **Gradient Boosting**, **AdaBoost**, and **Multi-Layer Perceptron** deliver solid performance, with **AdaBoost** and **Gradient Boosting** offering a great balance between accuracy and speed. Ultimately, ensemble methods like **Random Forest** and **Gradient Boosting** stand out as the most reliable, providing strong predictive power without compromising too much on runtime.

*\*Question #12: If we had to explain to someone what really drives peoples’ perception of wine quality, what would you say based on your findings in this homework (e.g. Q8) and previous ones? Are there 2-3 features we can say are consistently most important? If so, can you hypothesize why those features might be important (hint: do some googling)?*

***HW1: Decision Trees***

Selected features: ['volatile acidity', 'sulphates', 'alcohol']

Removed features: ['fixed acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH']

**Hw2 Random forest:**

*Removed features:*  *'fixed acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH'*

Selected features: *'volatile acidity', 'sulphates', 'alcohol'*

**HW3 Boosting:Binnig=0**

Selected ['volatile acidity', 'sulphates', 'alcohol']

Features (total/selected): 11 / 3

Features not selected: [ 'fixed acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH']

**HW4 SVM Q8:**

Selected: [‘volatile acidity’, ‘chlorides’, ‘density’, ‘sulphates’, ‘alcohol’]

Features (total/selected): 11 5

Removed: [‘fixed acidity’, ‘citric acid’, ‘residual sugar’, ‘free sulfur dioxide’, ‘total sulfur dioxide’, ‘pH’]

*Answer:*

*Based on the findings from this homework and previous ones, three features consistently emerge as the most important drivers of people's perception of wine quality:*

1. *Volatile Acidity*
2. *Sulphates*
3. *Alcohol*

*These features were consistently selected across multiple models (Decision Trees, Random Forest, Boosting, and SVM), indicating their strong influence on wine quality. Let's explore why these features might be so important:*

***Volatile acidity*** *(VA) is a measure of acids in wine that can indicate quality, spoilage, and bacterial infection. While some VA can add complexity, too much can make wine taste like vinegar or nail polish remover. Wines with lower VA are generally considered higher quality*

***Sulphates*** *are compounds that act as preservatives and antioxidants in wine, helping to stabilize it and prevent oxidation. Sulphates enhance a wine's longevity and freshness, which are often associated with higher quality. They also contribute to the perception of body and structure in wine, making it feel more robust and well-rounded. Proper sulphate levels are essential for maintaining flavor integrity over time.*

***Alcohol content:*** *Higher alcohol levels are often associated with riper grapes and fuller-bodied wines, which many consumers perceive as higher quality. Alcohol contributes to the wine's body, warmth, and can enhance flavor intensity*

*Additional features from HW4 Q8:*

*The* ***density*** *of the wine is an important parameter that is necessary to determine the extract. Wine is a mixture containing mainly dissolved solids (sugars, acids, phenols and mineral salts), which increase its density . On average, high quality wines have lower density than low and medium quality wines. However, density is also correlated with other factors that affect wine quality, such as sugar, acidity, and SO2 content.*

*Sodium* ***chloride*** *adds saltiness to wine, which can enhance or detract from the overall flavor.*