Cmpsc 448 HW 4

Problem 1) $f(w) = \|Xw - y\|_2^2 + \lambda\|w\|_2^2$ RR formula

a) expand the square norm

$\|Xw-y\|_2^2 = (Xw-y)^T(Xw-y) = w^TX^TXw - 2y^TXw + y^Ty$

so then $f(w) = w^TX^TXw - 2y^TXw + y^Ty + \lambda w^Tw$

take gradient:

$\nabla f(w) = 2X^TXw - 2X^Ty + 2\lambda w = 0$

$X^TXw + \lambda w = X^Ty$

$(X^TX + \lambda)w = X^Ty$

Since λ is a constant, we cant add it to matrix, so we can rewrite as λI

$(X^TX + \lambda I)w = X^Ty \longrightarrow w = (X^TX + \lambda I)^{-1}X^Ty$

To show that $X^TX + \lambda I$ is invertable, use SVD for X

$X = U\Sigma V^T$ where $U \in \mathbb{R}^{n\times n}$ $\Sigma \in \mathbb{R}^{n\times d}$ $V \in \mathbb{R}^{d\times d}$

$X^TX = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^TU^TU\Sigma V^T = V\Sigma^T\Sigma V^T$

Since U and V are orthogonal matrices, the eigenvalues of X^TX are $\sigma_1^2, \sigma_2^2 \dots \sigma_r^2$ which are the squared singular values of X then adding λI

$X^TX + \lambda I = V(\Sigma^T\Sigma + \lambda I)V^T$

then Since $\Sigma^T\Sigma$ is a diagonal matrix with entries $\sigma_i^2 + \lambda$ the squaring ensures positivity for the eigenvalues and adding λ wont change the sign since $\lambda > 0$ so therfore all eigenvalues are positive.

b) $X^TXw + \lambda Iw = X^Ty \longrightarrow \lambda w = X^Ty - X^TXw \rightarrow w = \frac{1}{\lambda}(X^Ty - X^TXw)$

$w = X^T\alpha$ plug into equation: $X^TX(X^T\alpha) + \lambda I(X^T\alpha) = X^Ty$

$X^T(XX^T\alpha) + I\lambda X^T\alpha = X^Ty$

$X^T(XX^T\alpha + I\lambda\alpha) = X^Ty \rightarrow XX^T\alpha + I\lambda\alpha = y$

$(XX^T + \lambda I)\alpha = y \rightarrow \alpha = (XX^T + \lambda I)^{-1}y$

so $w = X^T\alpha = X^T(XX^T + \lambda I)^{-1}y$

c) The span is the set of all possible linear combinations of the rows

To be in the span of the data, w must be in the span of X which is an nxd matrix α is nx1 so $X^T\alpha$ is dx1 so w is a linear combination of rows of X since it is a linear combination of rows, w is a subset of the span.

d) we have $(X^TX + \lambda I)w = X^Ty$

Substitute $w = X^T\alpha$

$$(X^TX + \lambda I)X^T\alpha = X^Ty \rightarrow X^TXX^T\alpha + \lambda X^T\alpha = X^Ty$$

$$XX^T\alpha + \lambda\alpha = y \rightarrow (XX^T + \lambda I)\alpha = y \rightarrow \alpha = (XX^T + \lambda I)^{-1}y$$

e) $Xw = X(X^T\alpha) = (XX^T)\alpha$

from previous part, we have $\alpha = (XX^T + \lambda I)^{-1}y$, plug in:

$$Xw = (XX^T)(XX^T + \lambda I)^{-1}y$$

f) $w_*^Tx = (X^T\alpha)^Tx = \alpha^TXx$ then substitute known α

$$= ((\lambda I + XX^T)^{-1}y)^TXx = y^T(\lambda I + XX^T)^{-1}Xx$$

Xx can be rewriting as x^TX^T can also rewrite given matrix as $K = XX^T$

$w_*^Tx = y^T(\lambda I + K)^{-1}k(x)$ where $k(x)$ is kernel function $k(x_i, x)$

g) The Gram matrix K, where $K_{ij} = k(x_i, x_j)$

we can use the Gaussian kernel: $k(x_i, x_j) = e^{\frac{-\|x_i - x_j\|^2}{2\sigma^2}}$

α will be $(\lambda I + K)^{-1}y$ so $w_*^Tx = y^T(\lambda I + K)^{-1}Xx$

Xx contains inner products with training data, can be replaced by a kernel function. $Xx = k(x_i, x)^T = k(x)^T$

The prediction formula: $k(x)^T(\lambda I + K)^{-1}y$

Problem 2)

a) if the color is yellow, predict edible weights originally: 1/16

correct predictions: ~~++++ ++++~~

Since there are 10 correct predictions, there are 6 incorrect classified

error: $\epsilon_1 = \frac{6}{16} = 0.375$

Strength: $\alpha_1 = \frac{1}{2}\log\left(\frac{1-0.375}{0.375}\right) = \frac{1}{2}\log\left(\frac{0.625}{0.375}\right) = \frac{1}{2}\log(1.667) \approx 0.2554$

b) adjust weights: $w_i^1 = \frac{1}{Z} w_i^0 e^{-\alpha_1 y_i f_1(x_i)}$

increase weights for points that are incorrectly classified $y_i f_1(x_i)$

D2 D3 D10 D12 D13 D14 are incorrectly classified, for these are -1

so we have $\frac{1}{16}e^{-0.2554}$ are the unnormalized weights for points that are correctly classified

$\frac{1}{16}e^{0.2554}$ for incorrectly classified points

$\frac{1}{16}e^{-0.2554} \approx 0.0484$ $\frac{1}{16}e^{0.2554} \approx$ 0.0807

now find Z for normalizing: $Z = 10(0.0484) + 6(0.0807) = 0.9682$

now divide weights by $\frac{1}{0.9682}$: correctly classified = 0.05 incorrectly classified: 0.0834

$w_1^2 = w_4^2 = w_5^2 = w_6^2 = w_7^2 = w_8^2 = w_9^2 = w_{11}^2 = w_{15}^2 = w_{16}^2 = 0.05$ $w_2^2 = w_3^1 = w_{10}^1 = w_{12}^1 = w_{13}^1 = w_{14}^1 = 0.0534$

c) yes, if the training error is 0, we will have early stopping. This is because when the training error is 0, the data will be perfectly classifed There is no need to update the weights further.

Homework 4 Code

Kush Lalwani

Load Data and Packages

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import RandomizedSearchCV

from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier

# load data in LibSVM sparse data format
X_train, y_train = load_svmlight_file("a9aTrain.txt")
X_test, y_test = load_svmlight_file("a9aTest.t")

#Convert the -1/1 labels to a 0/1 label since that is the expected
input for the classifiers I am using
y_train[y_train == -1] = 0
y_test[y_test == -1] = 0
```

Chosen Algorithms

Random Forest

Random Forest is an Ensemble learning method that uses multiple decision trees to classify data. Each decision tree is trained on a bootstrapped sample of the original dataset, so this is considered a Bagging algorithm. Each of these smaller decision tree also only uses a subset of the features. The model will then take a vote based on the smaller decision trees that were created, allowing the model to classify a test point. This algorithm avoids the regular decision tree's tendency to overfit the data. Random Forest tends to reduce variance and overfitting

XGBoost

XGBoost is also an ensemble learning method that utilizes weak learners, often decision trees. This is a Boosting algorithm, since it starts with weak learners and slowly builds off of them to create a strong learner. This algorithm also includes a regularization term to prevent against overfitting of the data. Boosting is iterative, if previous data were misclassified in the previous weak learner, the next model will try to put more importance on the previously misclassified points. XGBoost tends to reduce bias and prevent against underfitting.

Description of Training Methodology

First I will set a seed for the training, so that all of the code is exactly reproducable. Then I will fit the basic models with all of the default parameters to the training data, to get a baseline sense of the accuracy that can be achieved with the respective models. Once I have the baseline accuracy, I will run 10-fold cross validation to find the best possible parameter for the models. In order to do this, I will use the gridsearch function alongside the cross validation to test every single combination of paramters. I will first initialize a list that will be used for possible value of a parameter. This will save a lot of space as compared to manually writing validation code to test every single parameter combination which could involve many nested loops to test parameters. I will do this to train the model for both the random forest model as well as the decision trees.

Edit: After trying to run the code using cross validation with the grid search, it took too long to run. So I decided on an alternative approach, which was to use a random search to do my cross validation. Random searching will randomly sample a subset of possible parameter values rather than computing every single possible combination of parameters. This random searching will ensure that not too many models end up being trained, which will take too long to run. I also decided to opt for 3-fold cross validation instead of 10, which also significantly reduced the amount of time the code takes to run. Otherwise, my approach was the exact same except for the fact that I used random search instead of gridsearch.

```
#create a 80/20 training validation split for the baseline classifiers
np.random.seed(448)

seed = 448
test_size = 0.2
X_train_base, X_val_base, y_train_base, y_val_base =
train_test_split(X_train, y_train, test_size=test_size,
random_state=seed)
```

Random Forest Training

The Random Forest Classifier has the following parameters that are to be trained:

1. **n_estimators**: This parameter tunes the number of tree models that are trained, the default value for this parameter is `100`. This means that there are 100 decision trees trained in this algorithm by default.
2. **bootstrap**: This parameter is used to decide whether the algorithm uses bootstrapped samples of the whole dataset. Bootstrap samples are a random resampling of the dataset with replacement. By default the parameter is set to `True`; if it is set to `False`, it will use the entire dataset to train the model.
3. **max_depth**: This parameter tunes the maximum depth of each decision tree in the model. This by default, is set to `None`. If the model is trained with the default parameter, the tree will continue to expand until each of the nodes are pure; meaning that each leaf node will all contain the same class.
4. **min_impurity_decrease**: This parameter tunes how the nodes determine which feature to split on. The model will split on a proposed feature if the impurity score decrease of the model is greater than or equal to the set threshold. By default, the parameter is set to `0.0`.

5. **min_samples_leaf**: This parameter tunes the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. By default, the parameter is set to `1`.

```
baseline_rf = RandomForestClassifier(random_state=448) #set the random state because the random seed is not enough to ensure the reproducability
baseline_rf.fit(X_train_base,y_train_base)

y_pred_base_rf = baseline_rf.predict(X_val_base)
predictions = [round(value) for value in y_pred_base_rf]

# evaluate predictions
accuracy = accuracy_score(y_val_base, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 82.85%
```

Now that we have the baseline accuracy of 82.85%, we can run the cross validation using the grid. We will then see which combination of parameters will have the highest accuracy.

```
param_grid_rf = {
    'n_estimators': [50, 100, 200],  # Number of trees
    'bootstrap': [True, False],  # Whether to use bootstrapping
    'max_depth': [10, 20, 30, None],  # Maximum depth of trees
    'min_impurity_decrease': [0.0, 0.01, 0.05],  # Threshold for a node to split
    'min_samples_leaf': [1, 2, 5]  # Minimum samples per leaf node
}

rf = RandomForestClassifier(random_state=448)

random_search_rf = RandomizedSearchCV(
    estimator=rf,  # Random Forest model
    param_distributions=param_grid_rf,  # Parameter grid
    n_iter=10,  # Number of random samples to try
    cv=3,  # 3-fold cross-validation
    scoring='accuracy',
    n_jobs=-1,
    verbose=2,
    random_state=448
)

random_search_rf.fit(X_train, y_train)

print("Best Parameters:", random_search_rf.best_params_)
print("Best Cross-Validation Accuracy:", random_search_rf.best_score_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Parameters: {'n_estimators': 200, 'min_samples_leaf': 5,
```

```
'min_impurity_decrease': 0.0, 'max_depth': 20, 'bootstrap': False}
Best Cross-Validation Accuracy: 0.845305826941385
```

As we can see, the best parameters with our cross validation were found to be:

- n_estimators: 200
- bootstrap: False
- max_depth: 20
- min_impurity_decrease: 0.0
- min_samples_leaf: 5

And we found the accuracy with this set of parameters to be: 84.53% So then the Cross Validation error will be: 15.47%

Create the table of the hyperparameters

```
results_df = pd.DataFrame(random_search_rf.cv_results_)[
    ['param_n_estimators', 'param_max_depth',
'param_min_samples_leaf',
     'param_min_impurity_decrease', 'param_bootstrap',
'mean_test_score']
]

# Compute error rate
results_df['error_rate'] = 1 - results_df['mean_test_score']

# Sort by best accuracy
results_df = results_df.sort_values(by='mean_test_score',
ascending=False)

# Rename columns for clarity
results_df.rename(columns={'mean_test_score': 'cv_accuracy'},
inplace=True)

# Display table
print(results_df)

# Optionally, save the table to a CSV for your report
results_df.to_csv("rf_hyperparameter_results.csv", index=False)
```

```
   param_n_estimators  param_max_depth  param_min_samples_leaf  \
3                 200               20                       5
7                 100               10                       5
9                 100               30                       1
2                  50               30                       5
1                 100               30                       1
0                 100               20                       5
5                  50               30                       1
4                  50               20                       1
6                  50               20                       1
```

```
8                  100               10                          1

   param_min_impurity_decrease  param_bootstrap  cv_accuracy
error_rate
3                         0.00            False     0.845306
0.154694
7                         0.00            False     0.840177
0.159823
9                         0.00             True     0.834803
0.165197
2                         0.01             True     0.760572
0.239428
1                         0.01             True     0.759190
0.240810
0                         0.05             True     0.759190
0.240810
5                         0.05            False     0.759190
0.240810
4                         0.01            False     0.759190
0.240810
6                         0.05             True     0.759190
0.240810
8                         0.01            False     0.759190
0.240810
```

XGBoost Training

The XGBoost classifier has the following parameters that are to be trained:

1. **n_estimators**: This parameter is the amount of estimators that the model will use. So this will be the amount of iterations of boosting that the model will have. By default the parameter will take value `1`.
2. **max_depth**: This parameter tunes the depth of the decision trees that are used as the weak learners for this model. By default this parameter is set to `6`.
3. **reg_lambda**: This parameter tunes the lambda value of the L2 regularization parameter. By default, it is set to `1.0`.
4. **learning_rate**: This is the learning rate of the algorithm which is similar to the eta parameter for gradient descent. By default, the parameter is set to `0.3`.
5. **missing**: This parameter is the value in the data which needs to be present as a missing value. By default it is set to `numpy.nan`. It take a float.
6. **objective**: This parameter determines the learning task of the algorithm. You can also change the objective function that is being optimized. You can also set a custom objective function that will be optimized. It takes in a scikit learn objective function that should be optimized. By default it is set to a binary logistic classifer.

```
np.random.seed(448)

xgboost_base = XGBClassifier()
xgboost_base.fit(X_train_base,y_train_base)
```

```
y_pred_base_xgb = xgboost_base.predict(X_val_base)
predictions_xgb = [round(value) for value in y_pred_base_xgb]

# evaluate predictions
accuracy = accuracy_score(y_val_base, predictions_xgb)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 84.43%
```

As we can see the baseline accuracy is 84.43%, now we can optimize the parameters.

```
param_dist_xgb = {
    "n_estimators": [50, 100, 200],
    "max_depth": [3, 6, 10, None],
    "learning_rate": [0.01, 0.05, 0.1],
    "reg_lambda": [0, 0.1, 1],
    "missing": [np.nan],  # We dont actually want to tune this
parameter because it will just treat missing values as NaN
    "objective": ["binary:logistic"] # We also shouldn't change this
because we have a classification problem that we are dealing with
}

xgb = XGBClassifier(random_state=448)

xgb_search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_dist_xgb,
    n_iter=10,  # Try 10 random parameter combinations
    cv=3,  # 3-fold cross-validation
    n_jobs=-1,  # Use all available CPU cores
    verbose=2,  # Print progress
)

xgb_search.fit(X_train, y_train)

print("Best Parameters:", xgb_search.best_params_)
print("Best Cross-Validation Accuracy:", xgb_search.best_score_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Parameters: {'reg_lambda': 1, 'objective': 'binary:logistic',
'n_estimators': 200, 'missing': nan, 'max_depth': None,
'learning_rate': 0.05}
Best Cross-Validation Accuracy: 0.8495747649631169
```

As we can see, the best parameters with our cross validation were found to be:

- n_estimators: 200
- max_depth: None
- lambda: 1

- learning_rate: 0.05
- missing: np.nan
- objective: binary:logistic

And we found the accuracy with this set of parameters to be: 84.96% So then the Cross-Validation error will be 15.04%

Create the table of the hyperparameters

```
results_df = pd.DataFrame(random_search_rf.cv_results_)[
    ['param_n_estimators', 'param_max_depth',
'param_min_samples_leaf',
     'param_min_impurity_decrease', 'mean_test_score']
]

# Convert accuracy to error rate
results_df['error_rate'] = 1 - results_df['mean_test_score']

# Sort by best accuracy
results_df = results_df.sort_values(by='mean_test_score',
ascending=False)

# Display the table
print(results_df)
```

```
   param_n_estimators  param_max_depth  param_min_samples_leaf  \
3                 200               20                       5
7                 100               10                       5
9                 100               30                       1
2                  50               30                       5
1                 100               30                       1
0                 100               20                       5
5                  50               30                       1
4                  50               20                       1
6                  50               20                       1
8                 100               10                       1

   param_min_impurity_decrease  mean_test_score  error_rate
3                         0.00         0.845306    0.154694
7                         0.00         0.840177    0.159823
9                         0.00         0.834803    0.165197
2                         0.01         0.760572    0.239428
1                         0.01         0.759190    0.240810
0                         0.05         0.759190    0.240810
5                         0.05         0.759190    0.240810
4                         0.01         0.759190    0.240810
6                         0.05         0.759190    0.240810
8                         0.01         0.759190    0.240810
```

Run the model on the test data

```
# Get the best Random Forest model
best_rf = random_search_rf.best_estimator_

# Make predictions on the test set
y_pred_rf = best_rf.predict(X_test)

# Compute accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Test Accuracy: {accuracy_rf:.4f}")
```

```
Random Forest Test Accuracy: 0.8485
```

For the random forest model, the accuracy on the test set is 84.85% which is similar to the accuracy that we had for the training stage

```
best_xg = xgb_search.best_estimator_

# Make predictions on the test set
y_pred_xgb = best_xg.predict(X_test)

# Compute accuracy
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print(f"XGBoost Test Accuracy: {accuracy_xgb:.4f}")
```

```
XGBoost Test Accuracy: 0.8536
```

As we see here, the accuracy is 85.36% which is similar to the training accuracy as well.