CS 249 | HW #3 | Jason Chapman

(1) a) $f = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$ // Ridge Regression

$\Rightarrow f = (\underline{y} - \underline{X}\underline{\beta})^T (\underline{y} - \underline{X}\underline{\beta}) + \lambda \underline{\beta}^T \underline{\beta}$

where $\underline{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$, $\underline{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ x_{2,1} & x_{2,2} & & \\ \vdots & & \ddots & \\ x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix}$, $\underline{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$

b) $\nabla f = \frac{\partial f}{\partial \beta} = -\underline{y}^T \underline{X} - \underline{X}^T \underline{y} + 2\underline{X}^T \underline{X} \hat{\underline{\beta}} + 2\lambda \hat{\underline{\beta}} = 0$

// $(\underline{y}^T \underline{X})^T = \underline{y} \underline{X}^T$

$\Rightarrow \quad -2\underline{X}^T \underline{y} + 2\underline{X}^T \underline{X} \hat{\underline{\beta}} + 2\lambda \hat{\underline{\beta}} = 0$

c) $\hat{\underline{\beta}} (2\underline{X}^T \underline{X} + 2\lambda \underline{\underline{I}}) = 2\underline{X}^T \underline{y}$

$\Rightarrow \quad \hat{\underline{\beta}} = (\underline{X}^T \underline{X} + \lambda \underline{\underline{I}})^{-1} \underline{X}^T \underline{y}$

(2) a) After some exploratory analysis, I found that there were a few samples with multiple missing values, so I dropped the samples with 10+ missing values. Then, I simply replaced the missing value in the remaining samples with zeros because most missing values were on not key variables.

b) Categorical Features: Neighborhood, House Style

Predictive Features: Lot Area, Year Remodeled, Overall Quality, Overall Condition, Year Built, Full Bath, Total Room Above Ground, Garage Area, Garage Cars, Sale Price

c) See Attached

d.) See Attached

$RMSE_{training}$ = 22085.425 λ = 0.475

$RMSE_{validation}$ = 18616.368

- There actually appear to be some underfitting. This is simply because the chosen validation set fits the model better than the training set. // My analysis is correct, but I don't think I partitioned the data into training & validation sets to compute RMSE's following determination of the best regularization parameter.

e.) best k = 10

$RMSE_{training}$ = 42321.354

$RMSE_{cv}$ = 46357.323

- There is a large gap between the two RMSE's which indicates that there may be some overfitting.

f.) $RMSE_{Test}$ = 45366.023

(3) a) See Attached

b) I believe sex & age will be the most useful in prediction of what sort of people were more likely to survive the Titanic because women & children were prioritized. No, adding the ticket number as a feature is not a good idea. It does not offer any additional information to answer the question at hand. While it may be important to have a unique identifier for each sample, the passenger ID is sufficient to answer our question. Additionally, turning that categorical variable into a numerical variable would increase the size of the dataset significantly

c) See Attatched

Best subset of features: Pclass, Age, SibSp, Sex

d) AUC = 0.867325

(4)

4a) If I were building a prediction model for this problem, I would use a logistic regression model (binary classification), where Y=1 represents the outcome of one party winning and Y=0 represents the outcome of the same party losing (and thus the other party winning).

4b) Five features that would be important in predicting the outcome of an election would be the current majority party in the house of representatives, the current majority party in the senate, current president's approval rating, the history of each state's electoral votes for the last few elections, and an indicator number such as the Gallup ECI (Economic Confidence Interval).

4c) I would use the first option where I use each election as a sample in your dataset with features corresponding to both candidates. This way, the label will be 1 if the first candidate in the feature set wins, and 0 otherwise. This is primarily because this country tends to vote along party lines, and so the outcome of each election is dependent upon the state of the country at the time of the election (not dependent on the candidate themselves per say). So, it makes more sense to me to have each sample be a different election, where the label 1 if the candidate of one party wins and 0 otherwise. This also holds because in most US Presidential elections, only 2 parties have ever won the election.

4d) I would validate the performance of the model using cross-validation to randomly split the data into a training set and a validation set. Then, I would evaluate the RMSE of each to determine the performance of the model.

4e) If the error is small in the training set but large in the test set, then there is likely some overfitting in the model because it fits the current data well but does not generalize well to new points. If this was the case, I would focus on reducing the variance in the error. Gathering more data may help in this case but does not guarantee to fix the issue of overfitting. One way to fix this issue is to be selective with feature selection. Using either forward or backward selection here would likely help with the overfitting seen in this model.

4f) Gathering more election data in this case will not necessarily fit the issue of overfitting. Here, there is a problem with model complexity. The model complexity is too low, so to alleviate this problem I would try to increase the complexity of the model so that the model fits the data better.

4g) Most of the answers would stay the same, but you would have to change the prediction model used because the problem is no longer a binary classifier. Instead, could use a linear regression model where the outcome is percentage of electoral college votes for candidate A. Then, the difference between the vote share of candidates would be easy to calculate.

```
# CS 249 - HW #3
# Jason Chapman
import numpy as np
import pandas as pd
from scipy import stats
import statsmodels.api as sm
import matplotlib.pyplot as plt

2
df_train = pd.read_csv('train.csv')

# 2a
df_train.dropna(thresh=71, inplace=True)
df_train.fillna(value=0, inplace=True)

# 2c
X = df_train.drop('SalePrice', axis=1)
y = np.array(df_train.SalePrice)
X = pd.get_dummies(X)

# 2d
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
lambdas = np.logspace(-4, 3, 100)
rmse = []
for l in lambdas:
    model = Ridge(alpha=l, normalize=True)
    rmse.append(-1*np.mean(cross_val_score(model, X, y, scoring='neg_root_mean_sq
rmse = np.array(rmse)
ind = np.where(rmse == rmse.min())
param = lambdas[ind]

from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, shuffle=Tr
model = Ridge(alpha=param, normalize=True)
model.fit(X_train,y_train)
preds_train = model.predict(X_train)
model.fit(X_val,y_val)
preds_val = model.predict(X_val)
rmse_train = np.sqrt(np.mean((preds_train-y_train)**2))
rmse_val = np.sqrt(np.mean((preds_val-y_val)**2))

# 2e
from sklearn.neighbors import KNeighborsRegressor
k_neighbors = np.arange(1,25)
rmse_kn = []
for k in k_neighbors:
    model = KNeighborsRegressor(n_neighbors=k)
    rmse_kn.append(-1*np.mean(cross_val_score(model, X, y, scoring='neg_root_mean
rmse_kn = np.array(rmse_kn)
ind = np.where(rmse_kn == rmse_kn.min())
```

```
k_best = k_neighbors[ind]
rmse_cv = rmse_kn[ind]
model = KNeighborsRegressor(n_neighbors=int(k_best))
model.fit(X, y)
preds = model.predict(X)
rmse_kn_train = np.sqrt(np.mean((preds-y)**2))

# 2f
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=
model = Ridge(alpha=param, normalize=True)
model.fit(X_train,y_train)
preds = model.predict(X_test)
rmse_test = np.sqrt(np.mean((preds-y_test)**2))


# 3
titanic_train = pd.read_csv('train3.csv')
titanic_test = pd.read_csv('test3.csv')

# 3a
titanic_train['Age'].fillna(round(titanic_train['Age'].mean()), inplace=True)
titanic_train['Embarked'].fillna(method='bfill', inplace=True)
titanic_train.dropna(axis=1, inplace=True)
titanic_test['Age'].fillna(round(titanic_test['Age'].mean()), inplace=True)
titanic_test['Fare'].fillna(titanic_test['Fare'].mean(), inplace=True)
titanic_test['Embarked'].fillna(method='bfill', inplace=True)
titanic_test.dropna(axis=1, inplace=True)

# 3c
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn import metrics
X_titanic = titanic_train.drop('Survived', axis=1)
X_titanic = X_titanic.drop('Name', axis=1)
X_titanic = X_titanic.drop('Ticket', axis=1)
y_titanic = titanic_train.Survived
X_titanic = pd.get_dummies(X_titanic)
model = Ridge(alpha=0.5, normalize=True)
sfs = SequentialFeatureSelector(model).fit(X_titanic, y_titanic)
features = sfs.get_support()

# 3d
X_titanic = X_titanic.drop(['PassengerId', 'Parch', 'Fare', 'Embarked_C', 'Embark
X_train, X_test, y_train, y_test = train_test_split(X_titanic, y_titanic, test_si
model = Ridge(alpha=0.5, normalize=True).fit(X_train, y_train)
preds_test = model.predict(X_test)
fpr, tpr, thresholds = metrics.roc_curve(y_test, preds_test)
auc_test = metrics.auc(fpr, tpr)
```